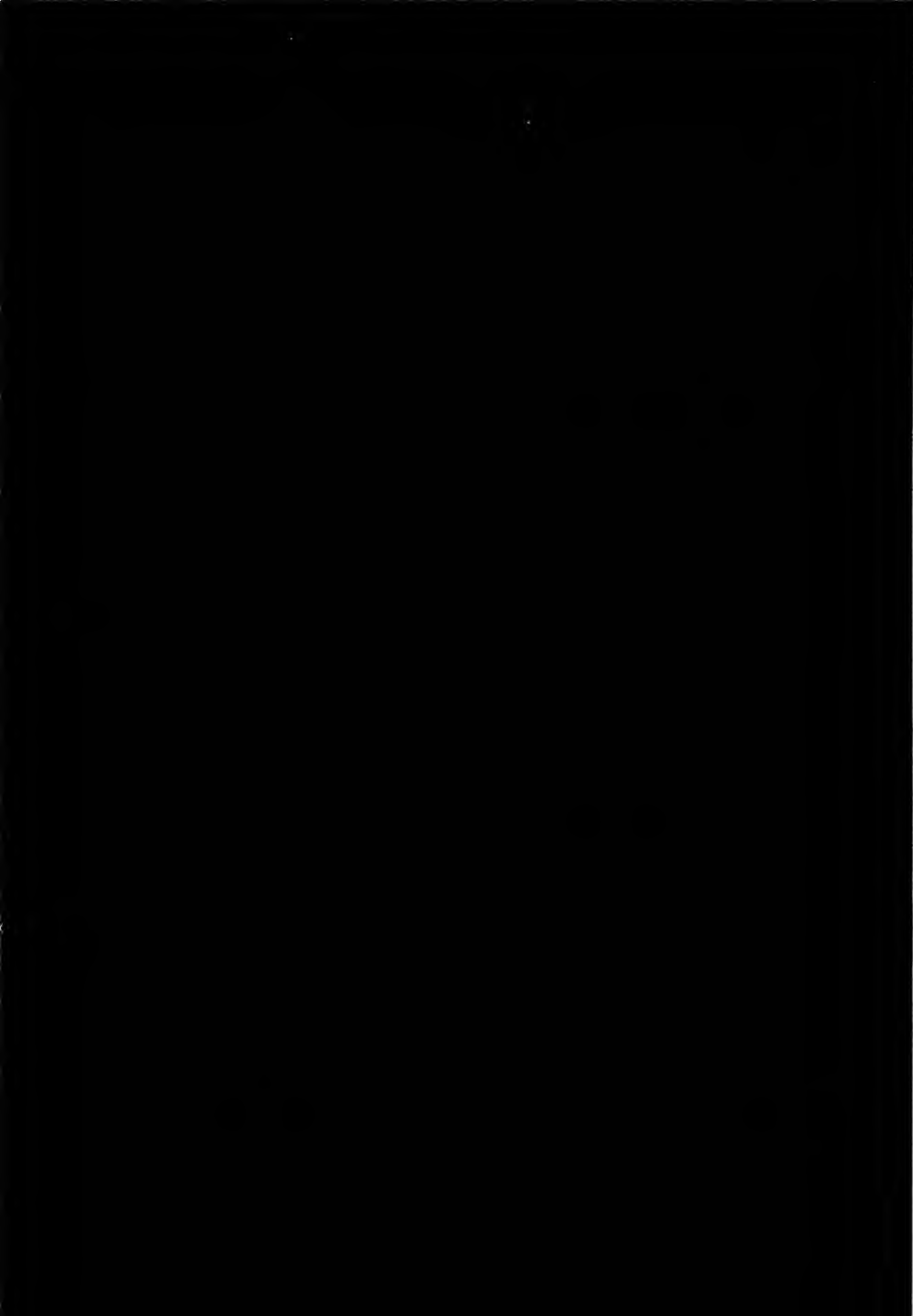
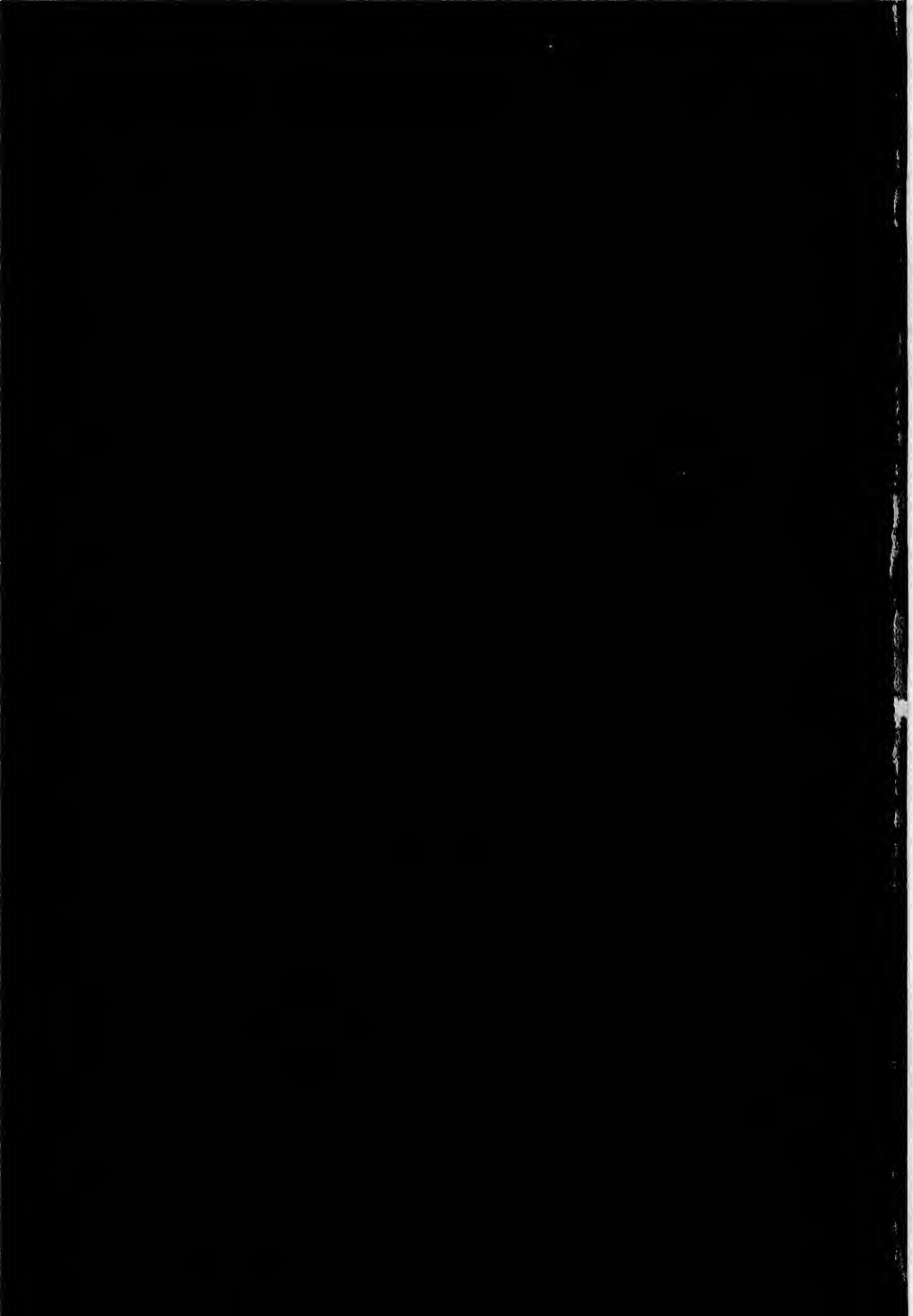
ソフトウェアツールの原点を探る

# はじめて読むアセンブラ

村瀬康治 著

アスキー出版局

Photographed by Akio Kambayashi

# はじめて読む
# アセンブラ

村瀬康治著

アスキー出版局

## 本書を読む前に

本書は，既刊『はじめて読むマシン語』で解説されている程度のコンピュータに対する基礎知識を持った読者を対象にした，Z-80および8080CPUのアセンブラと，ソフトウェア開発全般についての入門書です．

『はじめて読むマシン語』は，コンピュータの最も基礎的な知識である，メモリ，アドレス，2進数，16進数，ニーモニック，ハンドアセンブル，マシン語とCPU，スタック，プログラムの実行などについて，Z-80CPUを対象に，やさしく解説されています．これらの基礎知識をまだ学んでいない方は，まずコンピュータ全般の基礎学習から始めてください．

本書は，形式的には『はじめて読むマシン語』の続編になっていますが，『はじめて読むマシン語』そのものに，直接関係づけて書かれているわけではありません．よって，基礎学習は，任意の参考書を選んでもかまいません．

本書の実行例は，CP／M上での本格的なアセンブラやデバッガを使っていますが，各機種のBASIC上で，テープ・ベースの簡易アセンブラなどを使っても，十分な実習ができるように例題プログラムが作られています．本書のBASIC上の例題プログラムは，最も一般的な$N_{88}$-BASICでの実行例を示していますが，それぞれのBASIC上で実習する場合は，APPENDIX 1の各BASIC内ルーチンの違いについての注意書きをご覧ください．

# はじめに

### 8 ビット?

以前, アスキー出版局のスタッフと, 16 ビットで最も普及している 8086CPU の入門書, 例えば『はじめて読む 8086』などという本が, 書き得るかどうかについて話をしたことがあります. 私は, BASIC 言語を少し知っている程度で, アセンブラの知識のない人に向かって, 8086 アセンブラをどのように解説しても成功するとは思えません. つまり, 書き得ないという意見です.

もし私が書くとすれば, 最初に 8 ビットの 8080CPU について解説し, その知識を基にして 8086CPU の解説に接続していくような構成をとると思います. まず, 8 ビットのパーソナル・コンピュータ上で, コンピュータの仕組みと, アセンブラの基礎を学ぶことが, 何よりもたいせつであり, 効果的な学習法でもあるのです.

1 章でも述べるように, 実務用のパーソナル・コンピュータにおいて, 8 ビットの時代はすでに過ぎました. しかし, 8 ビットのマイクロコンピュータは, 今後とも多くの応用分野で大量に使われ続けるでしょう. そして何よりもコンピュータの学習に関しては, この 8 ビット・マシンを基にすることが鍵なのです.

### アセンブラ?

本書はアセンブラの入門書です. とはいっても, 私たちは日常, この「アセンブラ」という言葉の意味を, かなりあいまいに使っています. 例えば,

「このプログラムはアセンブラで書かれている」
「BASICよりアセンブラの方が速い」
「アセンブラはわからない」
「CP/M の8080アセンブラを使っている」

というような調子です．

「アセンブラ」とは本来，「アセンブルを行う道具」のことであり，アセンブリ言語で書かれたプログラムをマシン語に翻訳し，CPUが実行可能なマシン語のプログラムを生成する道具のことを指します．つまりアセンブラは，「マシン語プログラム生成ソフトウェア」なのです．言い換えれば，アセンブラは，ソフトウェア開発ツールと呼ばれるソフトウェアを開発するためのいくつかの「道具」のうち，アセンブリ言語のプログラムから，マシン語のプログラムを作り出すための，ソフトウェアでできたひとつの「道具」であるわけです．

ところが，日常の会話などでは，私もそうですが，「アセンブリ言語」，「アセンブラ」，「マシン語のプログラム」などを混同して，これらをアセンブラという言葉で代用しています．

本書は，ニーモニックや，そのプログラミング技法などの解説書ではありません．本書の主題は，本格的なソフトウェア開発を行う場合の全過程と，その作業の中心となる，アセンブラという道具を使うための基礎的な解説で

す．本書に登場する例題プログラムは，たった3種類で，しかもやさしいものばかりです．この3つのプログラムを基に，本格的なソフトウェア開発ツールを使った実習解説をしています．

80系の8ビットCPUのソフトウェア開発は，ザイログ社やインテル社の開発専用マシンなどを利用する場合を除いて，一般的にはCP／Mマシンに頼らざるを得ません．現実的に，CP／Mを抜きにして，本格的なソフトウェア開発を語ることはできないのです．本書でも実習の多くにCP／Mとその上で利用できるソフトウェア開発ツールを使っていますが，CP／Mについて多くのページを割くことはできませんでした．アセンブラを本格的に学習される方は，本書と並行して，別の参考書でCP／Mの使い方を学んでください（巻末の参考文献参照）．

冒頭でも述べたように，8ビット・マシンを基にした，ここでの基礎学習は，将来，16ビットや，32ビットのCPUを学ぶ場合に，非常に大きな力と意味を持つことになるはずです．本書を手にした今の気持ちを忘れずに，この機会を逃さずしっかりとした基礎作りをしてください．

「はじめに」で言うのも何ですが，本書の「あとがき」も，実はなるべく早いうちに読んでおいてほしいのです．

1984年　師走

著者　村瀬康治

# 目　次

イラスト　伝　陽一郎

# 1

# ソフトウェアの原点 アセンブラ

8ビット・パーソナル・コンピュータを前に本書を手にした方は，アセンブラを理解するほんとうによいチャンスです．既刊『はじめて読むマシン語』などで得られたコンピュータの基礎知識を基に，ソフトウェアの原点であり，母であるアセンブラの基礎をぜひ習得してください．

本章ではまず，8ビットのコンピュータとそのアセンブラに関する知識が，今後ともたいへん重要であり続けることの話から始めましょう．

Z-80や，8080，8085などのCPUを使った8ビットマシンは，内部の働きがコンピュータの基本原理に近く素直であるため，コンピュータの動作原理や，プログラミング技術を，基本からしっかりと理解するためには最も適しています．

コンピュータをほんとうに理解するには，CPUの働きやアセンブラの知識が必要不可欠であり，またこれらを学ぶことこそ，コンピュータ・アーキテクチャを理解するための近道です．そしてこの学習には，8ビットのコンピュータから入門することが最も効果的なのです．

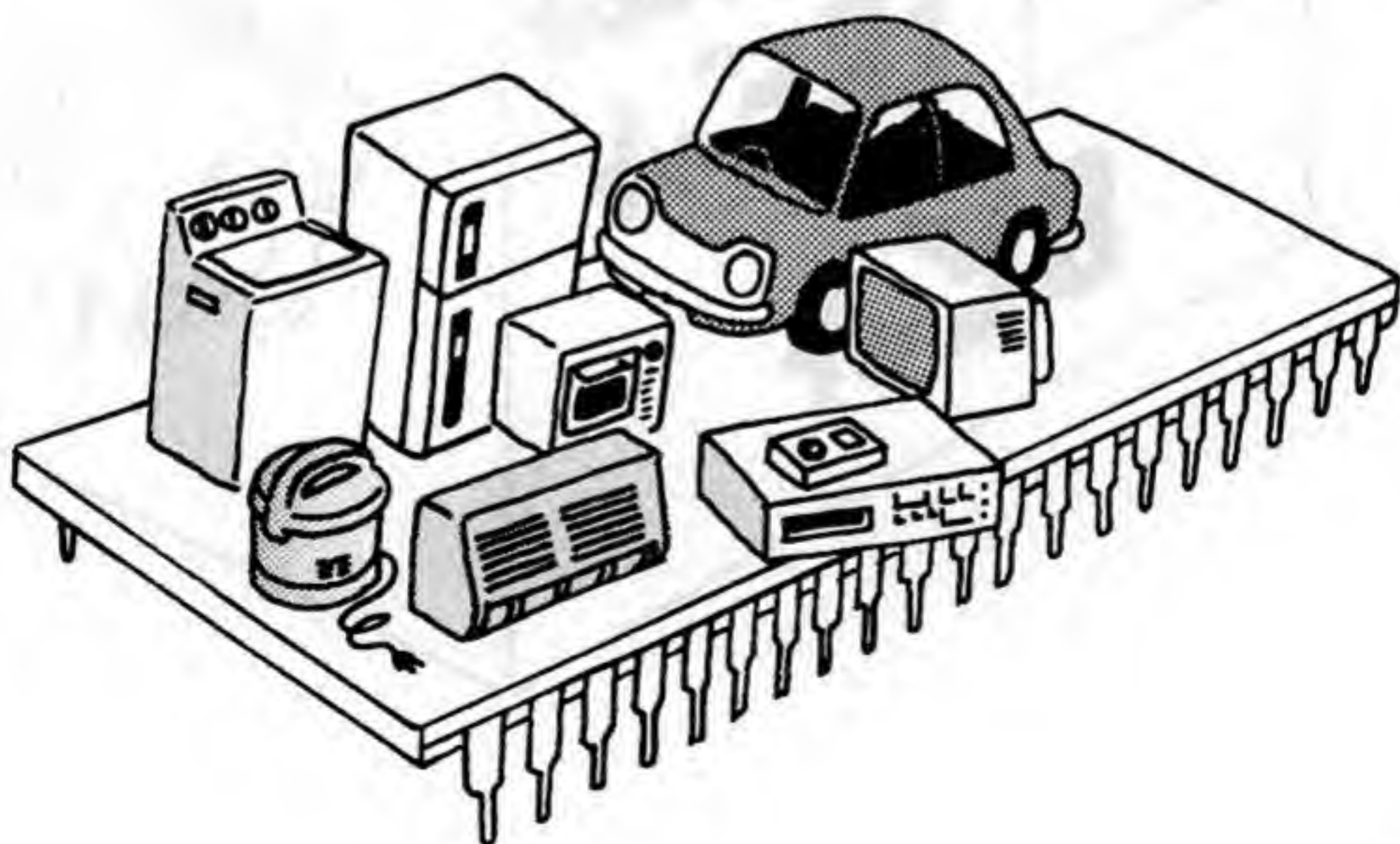
16ビットのCPUや，そのアセンブラは，8ビットのものに比べて格段に難しいものとなっています．初めてコンピュータの基礎を学ぶ人は，ぜひ8ビットから入門して，16ビットへと進むことをお勧めします．また，パーソナル・コンピュータ上での，各種の高級言語を学ぶ人も，8ビットCPU程度のアセンブラの知識は，基礎知識として身に付けた上で学習を始めるべきでしょう．

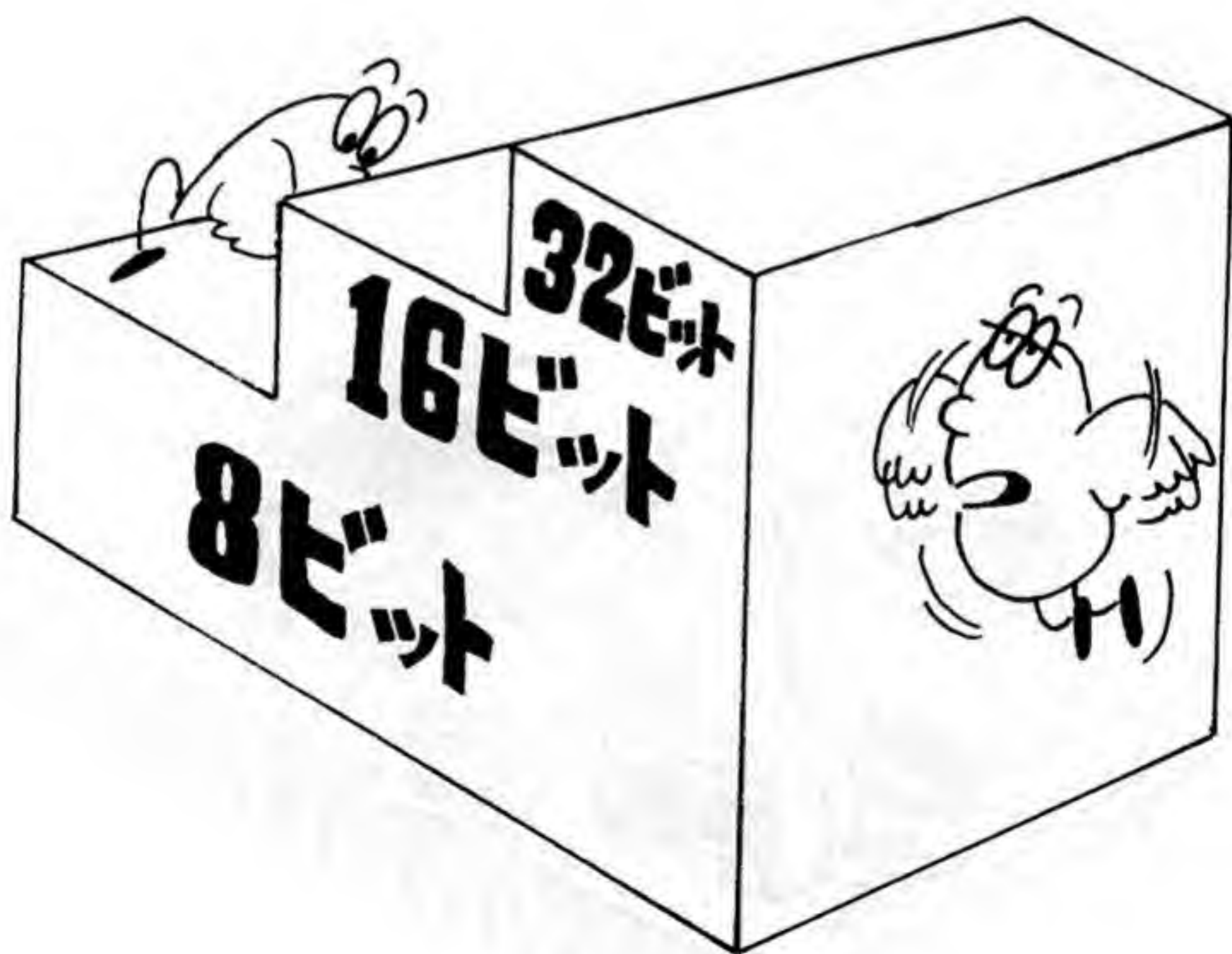
あなたの前の小さな8ビット・パーソナル・コンピュータ，この存在は，将来あなたが超大型のスーパー・コンピュータを扱うようになったとしても，それ以上に大きいものとなるに違いありません．

# 1/1 永遠の8ビットCPU

コンピュータの世界の進歩は著しく，パーソナル・コンピュータにおいても実務機は8ビットマシンから16ビットマシンに移り，さらに32ビットへと発展していこうとしています．しかし，8ビットのCPU，それもZ-80や8085などは，世の中のコンピュータの情勢がどうなろうとも，ずっとずっと永く使われ続けていくでしょう．

それは，8ビットCPUが，ハードウェア的にもソフトウェア的にも，たいへん扱いやすい上に，適度な能力を備えているからです．つまり，非常に広範囲に応用できるCPUなのです．

マイクロプロセッサ(Z-80などのCPUのこと)の応用範囲は，パーソナル・コンピュータに限られているわけではありません．家庭電気製品や各種の機器の制御用に，人目に触れず組み込まれているものの方が圧倒的に多いのです．そしてそれらのCPUの能力は，8ビットやそれ以下の4ビットのもので十分な場合が多いのです．

8ビットCPUは，大量にかつ広範囲に使われています．ということは，マイクロコンピュータを応用したシステムを設計したり，そのソフトウェアを開発したりする人達が大勢必要になるわけです．しかし，こうした技術者の数は現在でも不足しており，近い将来にはさらに深刻な事態になると予想されています．BASICゲーム少年は大勢いても，その次のレベルに踏み込んでいく人は，意外と少ないのです．

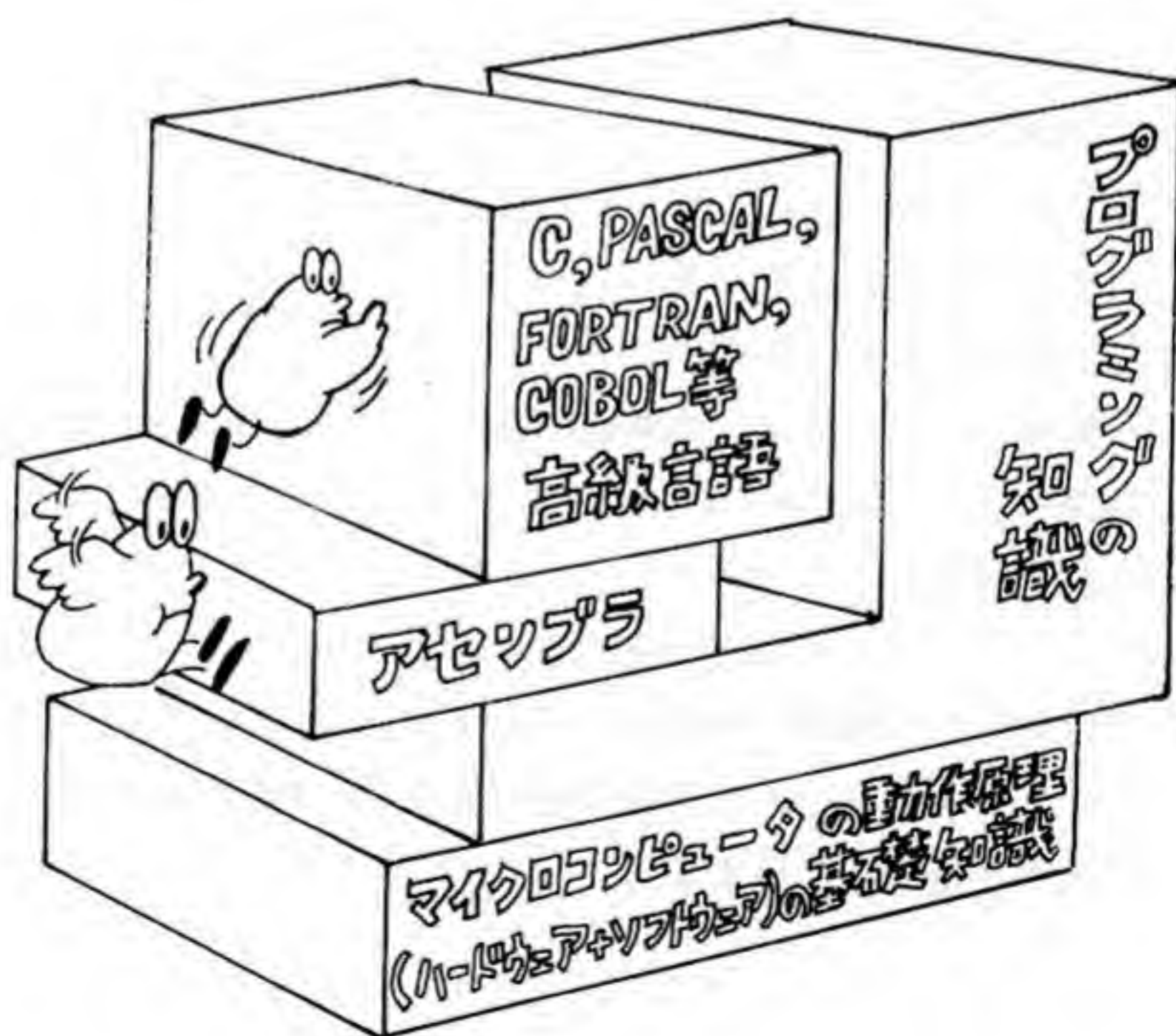
8ビットCPUは，今後もずっと使われ続けます．たとえ世の中が32ビットのスーパー・パーソナル・コンピュータの時代になったとしても，Z-80や8085などの8ビットCPUが，非常に重要な位置にあることに変わりはありません．

# 1/2 ソフトウェアの原点 アセンブラ

パーソナル・コンピュータのソフトウェアというと，すぐにBASICを思い浮かべる方も多いことでしょう．しかし，世の中の実用的なソフトウェアには，BASICはほとんど使われていません．前節で述べた，大量かつ広範囲に使われている4ビットや8ビットCPUを応用したシステム——例えば洗濯機「マイコンからまん棒」に，24Kとか，32KとかのBASICのROMが組み込まれているはずはないのです．

機器制御用のソフトウェアはもとより，パーソナル・コンピュータ上のビジネスソフトなども，本格的なものはほとんどがアセンブラかもしくはC(シー)，Pascal(パスカル)，COBOL(コボル)などのプログラミング言語によって書かれています．

グラフィックスのソフトも，本格的なものはBASICを使いません．特に高速な処理を必要とする場合は，グラフィックスを制御するためのハードウェアや，VRAM(グラフィックや文字を表示するためのメモリ，ビデオRAM)をアセンブラなどで作成したマシン語で直接コントロールしなければ，実用的なソフトウェアにはなりません．

米国に何年か遅れて日本のパーソナル・コンピュータ・シーンも，その本格的な利用やソフトウェアの開発には，8ビットではCP／M(シーピーエム)，16ビットではMS-DOS(エムエスドス)などのOS(オペレーティング・システム)を使うことが常識的になりました．同時に，パーソナル・コンピュータのメーカーやソフトウェア・ハウスのソフトウェア技術が向上するに伴い，アセンブラや各種のプログラミング言語による開発が常識化しています．またプログラミング言語自身も，ソフトウェア・メーカーの開発競争が盛んで，効率のよいものや使いやすいものが次々と出現しています．

BASIC 言語はやはりその名のとおり，ビギナー(Beginner)のための言語であることが，ますます明確になってきています．本書を手にされたみなさんはすでに自覚していることと思いますが，BASIC 言語は，学習用や家庭用または小規模で簡単な処理に使うための言語であることを，はっきりと認識しなければなりません．本格的なソフトウェアの開発を志す人は，アセンブラや先に挙げたプログラミング言語を使いこなすことが絶対的な条件になっています．そしてこれらの基礎はアセンブラにあるのです．

# 1/3 パソコン・ハッカー

「もうこれからは高級言語だよ，いまさらアセンブラなんて」と言う人もいるようですが，たぶんその人はマイクロコンピュータのソフトウェア開発の専門家ではないでしょう．大型汎用コンピュータの世界では，事務処理ソフトを開発しているプログラマーであれば，COBOLだけしか知らないとか，科学技術計算の分野であれば，FORTRANだけしか知らないという人も多くいます．このような専業化は，大型コンピュータのソフトウェアを生産する業界の仕組みによるものです．簡単にいうと，生産性の面から設計者，プログラマー，キーパンチャー，ユーザーといった分業化が進み，その結果パーソナル・コンピュータ上のソフトウェアのように，きめの細かいものは作っていられないようになっているのです．

ところがパーソナル・コンピュータを含むマイクロコンピュータの世界ではそうはいきません．少ない容量のメインメモリ，低い能力のCPUなど，ハードウェア上の大きな制約があります．大型機の場合のようにその強力なパワーに物を言わせて，ソフトウェアの作り方を何から何まで定型化してしまい，型どおりのやり方で押し切ることはできません．

パーソナル・コンピュータやマイクロコンピュータのソフトウェアは，きめ細かく，効率のよいものでなくてはなりません．よって，その技術者は，大型機の場合のような，専業化されたCOBOLプログラマー的なものではだめなのです．ハードウェア，OS，アセンブラを始め各種言語などの，幅広い知識を必要とし，それらを総合的に応用できる高度なソフトウェア技術が要求されます．

このような知識を持つ，ミニコンや，マイクロコンピュータのソフトウェア技術者を「ハッカー」と呼びます．

**ハッカー：Hacker**

この新語の持つイメージは，組織化，定型化されたプログラマー集団には属さず，Tシャツに長髪という，どちらかというときたない風体*で，一定した勤務時間はなく，いつの間にかどこからともなく仕事場に現れて一心にキーを叩く，高給優遇された高度ソフトウェアの開発者，というところでしょうか．

今，日本にも大勢のハッカーが活躍しています．ワードプロセッサの「JS-WORD」を書いたのもひとりのハッカー，パーソナルCADシステムの「CANDY」を書いたのもひとりのハッカー，プログラミング言語の「Prolog-J」や「LSI C」，「Rgy FORTH」**を書いたのもそれぞれひとりのハッカーです．そして彼らのソフトウェア技術は，ほとんど例外なく，アセンブラの知識を基にして，その上で各種の言語が使いこなされているのです．

コンピュータの基礎は8ビット・マシンであり，ソフトウェアの基礎はアセンブラです．今Z-80や8085CPUを持つパーソナル・コンピュータに向かってアセンブラを学ぼうとしている読者は，この両者にアプローチする最も有利な出発点に立っています．この機会を逃がしてはいけません．本書を片手に，一つひとつしっかりと読み進んでいってください．『はじめて読むマシン語』でも実感できたように，マシン語もアセンブラも，その一つひとつは実に単純なのです．

＊このイメージは当てにならない．私の知っているハッカー達は，なぜかキマッている人が多い．
＊＊これらは日本のソフトウェア・メーカーにより開発された，優良なソフトウェアである．

# 2

# アセンブラの全体像の把握

本章では、アセンブラとは何か、それはどのようなものか、あらかじめその大まかな全体像をつかんでおくための解説をしましょう．

『はじめて読むマシン語』でも強調しているように，Z-80や8085などのCPUに直接命令することができるのは，一般的に「マシン語」と呼ばれているものだけです．実際のプログラミングは，BASICやその他の高級言語を使って行われますが，最終的にはマシン語に変換されたものが実行されます．

マシン語は，何もないところから直接に作り出されるわけではなく，一般に各種の言語から生成されるものです．つまり，コンピュータを働かすソフトウェア（最終的にはマシン語）は，言語によってプログラミングされるのです．

12章で概説しますが，言語にはいろいろな種類や形態のものがあり，それぞれ特長を持っています．その中でもアセンブリ言語だけは特別な言語であり，CPUを直接コントロールし，CPUが持つすべての機能を最小の命令語単位で生成することが可能な，ただひとつの言語です．つまりアセンブリ言語は，CPUを働かすための最も基本的な言語であるわけです．

私たちは普通，アセンブリ言語のことをただの「アセンブラ」と呼んでいますが，このアセンブラは，プログラミングの基本言語として，マイクロコンピュータ，ミニコンピュータ，大型コンピュータなどの種類を問わず，すべてのコンピュータのCPU別にそれぞれ必ず用意されています．何はなくてもアセンブラだけはあるのです．その中で，Z-80CPUを対象にしたものが，本書で解説する「Z-80アセンブラ」なのです．

# 2/1 CPUに密着した言語 アセンブラ

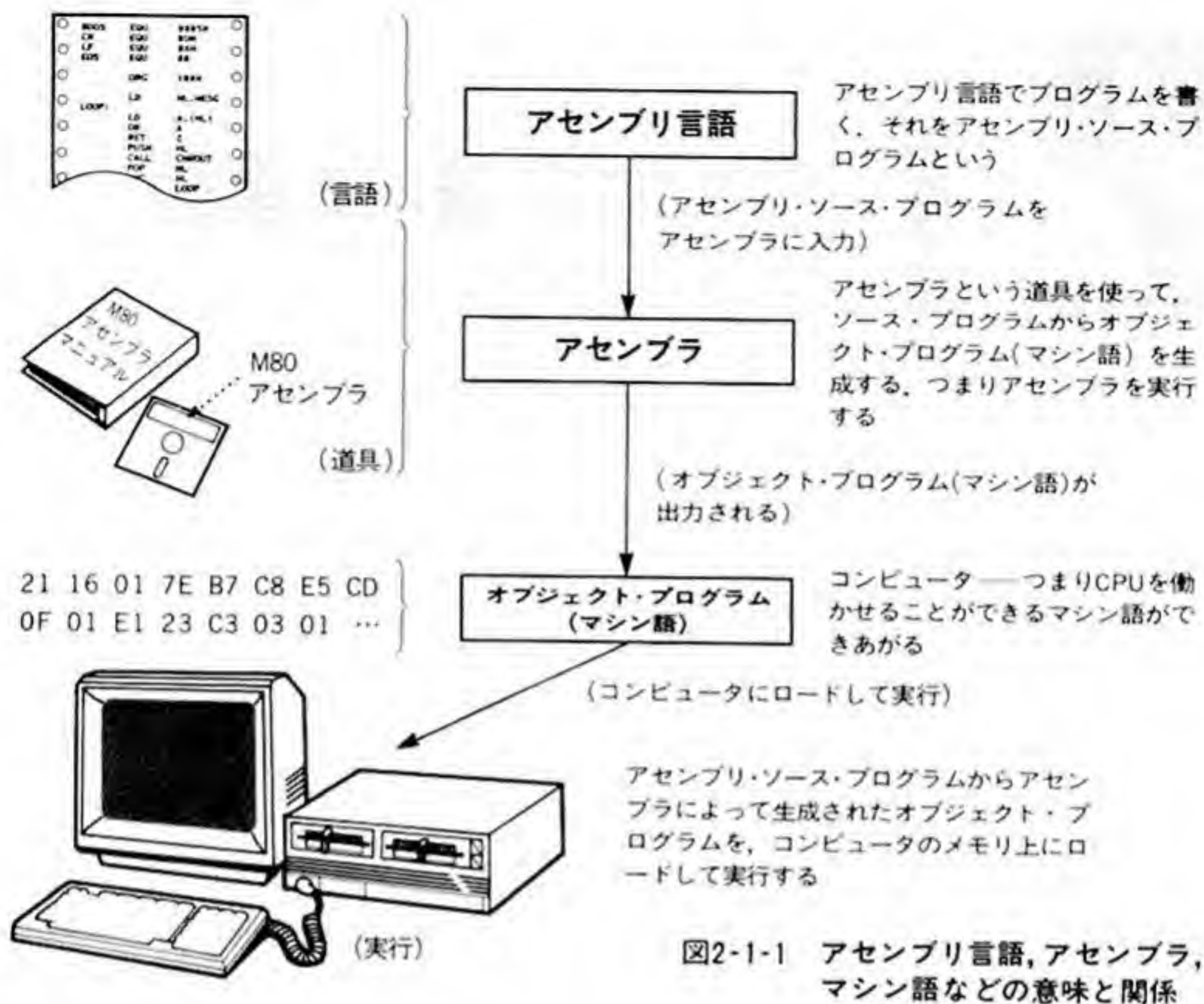
CPUのすべての働きを直接記述できる言語はアセンブラだけです．これは，アセンブリ言語の命令語の一つひとつが，マシン語(マシンコード)と1対1で対応しているためであり，これ以上CPUに密着した言語は存在しません．つまり，CPUを働かせるためのマシンコードを表現する各種の命令語をうまく組み合わせて，目的の仕事をさせるように記述するものが，アセンブリ言語であるわけです．

本章の冒頭でも述べましたが，マシン語——つまり，CPUを働かせる最終形態であるオブジェクト・プログラムはマシン「語」と呼ばれていますが，これは各種の言語から生成される「データ」と考えるのが適切です．マシン「語」というよりは，「マシンコード」とか「マシンデータ」とかいった方が混乱も少ないでしょう．

ではここで，まだそれぞれの意味が漠然としている，アセンブラ，アセンブリ言語，マシン語，オブジェクト・プログラムなどの言葉が，いったい何を表し，それらは何をするものか次ページの図で示します．

私たちは普通，アセンブリ言語のことも単に「アセンブラ」と呼んでいますが，本来アセンブラは，「アセンブルするもの」のことであり，アセンブリ言語で書かれたソース・プログラムを入力し，マシン語(オブジェクト・プログラム)を生成して出力するソフトウェアの「道具」(ソフトウェアツール)のことを指します．

- **アセンブリ言語**——**言語**（各種のプログラミング言語のひとつ）
- **アセンブラ**——**道具**（ソフトウェア製品で，それぞれのCPU用のものを使用する）

図2-1-1　アセンブリ言語，アセンブラ，マシン語などの意味と関係

普通の会話などでは，これらは区別されずに混同して使われており，また本書でも，アセンブリ言語のことを単にアセンブラと呼んでいる所もありますが，この2つの本来の意味は正しく区別して理解しておいてください．

## ソース・プログラムを見る

さて次は，アセンブリ言語によるプログラムの実例です．例題として示すのは，「Good Morning」というメッセージをCRTディスプレイに表示するだけの，アセンブラ(正しくはアセンブリ言語)で書かれた短い1つのプログラムです．正確には，

(復帰)　(改行)
Good＿Morning
(復帰)　(改行)

というように，復帰と改行が前後に入ります．このプログラムは，BASIC 言語であれば，ただ1行のプログラム，

PRINT "Good Morning"

に相当します．*

図2-1-2　アセンブリ・ソース・プログラム

```
;---------------------------;
;     MESSAGE OUT PROGRAM    ;
;---------------------------;
;
BDOS    EQU     0005H    ; system call entry point
CR      EQU     0DH      ; Carriage Return code
LF      EQU     0AH      ; Line Feed code
EOS     EQU     00       ; End Of String code
;
        ORG     100H     ; start address = 100H
;
        LD      HL,MESG  ; get top address of message
LOOP:                    ;          HL=character pointer
        LD      A,(HL)   ; get character code to output
        OR      A        ; end of string? (A=00?)
        RET     Z        ; if 'Z'=1, end of this program
        PUSH    HL       ; save character pointer
        CALL    CHROUT   ; character out
        POP     HL       ; restore character pointer
        INC     HL       ; pointer goes up
        JP      LOOP     ; jump for next character
;
;------ 1 character out subroutine ------
;
CHROUT:
        LD      C,2      ; CP/M system call
        LD      E,A      ;          console out
        CALL    BDOS     ;                function
        RET
;
;------ string data area for message ------
;
MESG:   DB      CR,LF,'Good Morning',CR,LF,EOS
;
        END              ; list end
```

このリストに書かれているのが，アセンブリ言語によるプログラムで，この作成には何らかのエディタ（プログラムの文章ファイルを作成するためのソ

* 正確には，PRINT : PRINT "Good Morning"．

フトウェアツール)を使います．このようなリストは，アセンブリ言語で書かれた，マシン語の素(もと)(Source)になるプログラムという意味で，「アセンブリ・ソース・プログラム」と呼ばれます．このソース・プログラムを，ソース・プログラムからマシン語を生成するための道具であるアセンブラに入力して，アセンブラを実行する――つまりソース・プログラムをアセンブルすると，CPUを直接コントロールし，実際にコンピュータを働かすマシン語(ソース・プログラムに対して，オブジェクト・プログラムという)が生成されます．同時に「アセンブルリスト」と呼ばれる，先のソース・プログラムに，生成されたマシン語やそのロード・アドレスなどが書き込まれたリストも出力されます．

ソース・プログラムの書き方や，アセンブラの実行のしかた，オブジェクト・プログラムの形式などについては，次章以降で解説しますので，ここでは途中を省略し，アセンブル後の結果を示すアセンブルリストを次に示します．先のソース・プログラムとよく比較してください．*

このアセンブルリストは，アセンブル作業によって生成されたオブジェクト・プログラムと，そのメモリ上へのロード・アドレスなどが，これらを生成する素になったアセンブリ・ソース・プログラムに付加された形式でタイプアウトされています．よってこのリストには，アセンブラの作業の全体的な姿がよく表れています．

ところで，このリストを見て，何だかゴチャゴチャしているな，と思われた方がいるかもしれません．それはきっと「コメント」と呼ばれる注釈文が多く書かれているためでしょう．後ほど，このコメントの部分を取り除き，もっとよく「中身」が見えるようにしたものを示しますが，まずその前に，アセンブリ・ソース・プログラムの「行」と，「コメント」についての，アセンブラの基本的な機能を2つ覚えておいてください．

- アセンブラはソース・プログラムを行単位でマシンコードに置き換える
- 各行のセミコロン[；]の右側を，アセンブラは無視する

* これはZ-80 CPU用のアセンブラの例だが，8080CPUのニーモニックを使ったものは4章で作成しているので必要があれば参照するとよい．

**図2-1-3　アセンブル結果のリスト**

```
                       ;-----------------------------;
                       ;      MESSAGE OUT PROGRAM     ;
                       ;-----------------------------;
                       ;
      (0005)           BDOS    EQU     0005H    ; system call entry point
      (000D)           CR      EQU     0DH      ; Carriage Return code
      (000A)           LF      EQU     0AH      ; Line Feed code
      (0000)           EOS     EQU     00       ; End Of String code
                       ;
                               ORG     100H     ; start address = 100H
                       ;
0100  211601                   LD      HL,MESG  ; get top address of message
                       LOOP:                    ;          HL=character pointer
0103  7E                       LD      A,(HL)   ; get character code to output
0104  B7                       OR      A        ; end of string? (A=00?)
0105  C8                       RET     Z        ; if 'Z'=1, end of this program
0106  E5                       PUSH    HL       ; save character pointer
0107  CD0F01                   CALL    CHROUT   ; character out
010A  E1                       POP     HL       ; restore character pointer
010B  23                       INC     HL       ; pointer goes up
010C  C30301                   JP      LOOP     ; jump for next character
                       ;
                       ;------ 1 character out subroutine ------
                       ;
                       CHROUT:
010F  0E02                     LD      C,2      ; CP/M system call
0111  5F                       LD      E,A      ;          console out
0112  CD0500                   CALL    BDOS     ;                 function
0115  C9                       RET
                       ;
                       ;------ string data area for message ------
                       ;
0116  0D0A476F         MESG:   DB      CR,LF,'Good Morning',CR,LF,EOS
      6F64204D
      6F726E69
      6E670D0A
      00
                       ;
0127                           END              ; list end
```

ロード・アドレス　　オブジェクト・プログラム　　アセンブリ・ソース・プログラム

「行単位」とは，CRTディスプレイやリストの左端から，リターンキーが入力されるまでの入力文字列のことです．リターンキーが入力されなければ，CRTディスプレイやプリンタの1行の文字数の制約上から複数行に表示されたとしても，それは「1行」です．

「コメント」については，各行の[；]の右側には何を書いてもアセンブラはそれを無視し，影響を受けません．よって，先のリストの例のように，標題や注釈，または1行空けるために[；]が使われます．

では，**図2－1－2**のアセンブリ・ソース・プログラムから，すべてのコメント類を省いたものを**図2－1－4**に示します．省いた部分は，すべて無視される部分ですから，アセンブラから見れば，ソース・プログラムの内容の変化はまったくないのと同じです．

**図2-1-4　すべてのコメント類を省いたソース・プログラム**

```
BDOS    EQU     0005H
CR      EQU     0DH
LF      EQU     0AH
EOS     EQU     00

        ORG     100H

        LD      HL,MESG
LOOP:
        LD      A,(HL)
        OR      A
        RET     Z
        PUSH    HL
        CALL    CHROUT
        POP     HL
        INC     HL
        JP      LOOP

CHROUT:
        LD      C,2
        LD      E,A
        CALL    BDOS
        RET

MESG:   DB      CR,LF,'Good Morning',CR,LF,EOS

        END
```

これで，ずいぶんスッキリしました．アセンブラにとって，先のソース・プログラムの処理の対象となる内容は，これだけであったわけです．このソース・プログラムをアセンブルすると，当然ですが，**図2－1－1**のソース・プログラムをアセンブルした場合と，まったく同一のオブジェクト・プログラムが生成されます．

その結果のアセンブルリストを**図2－1－5**に示しますが，これも当然のことながら，**図2－1－2**のアセンブルリストからコメント類を省いたものと同一になります．

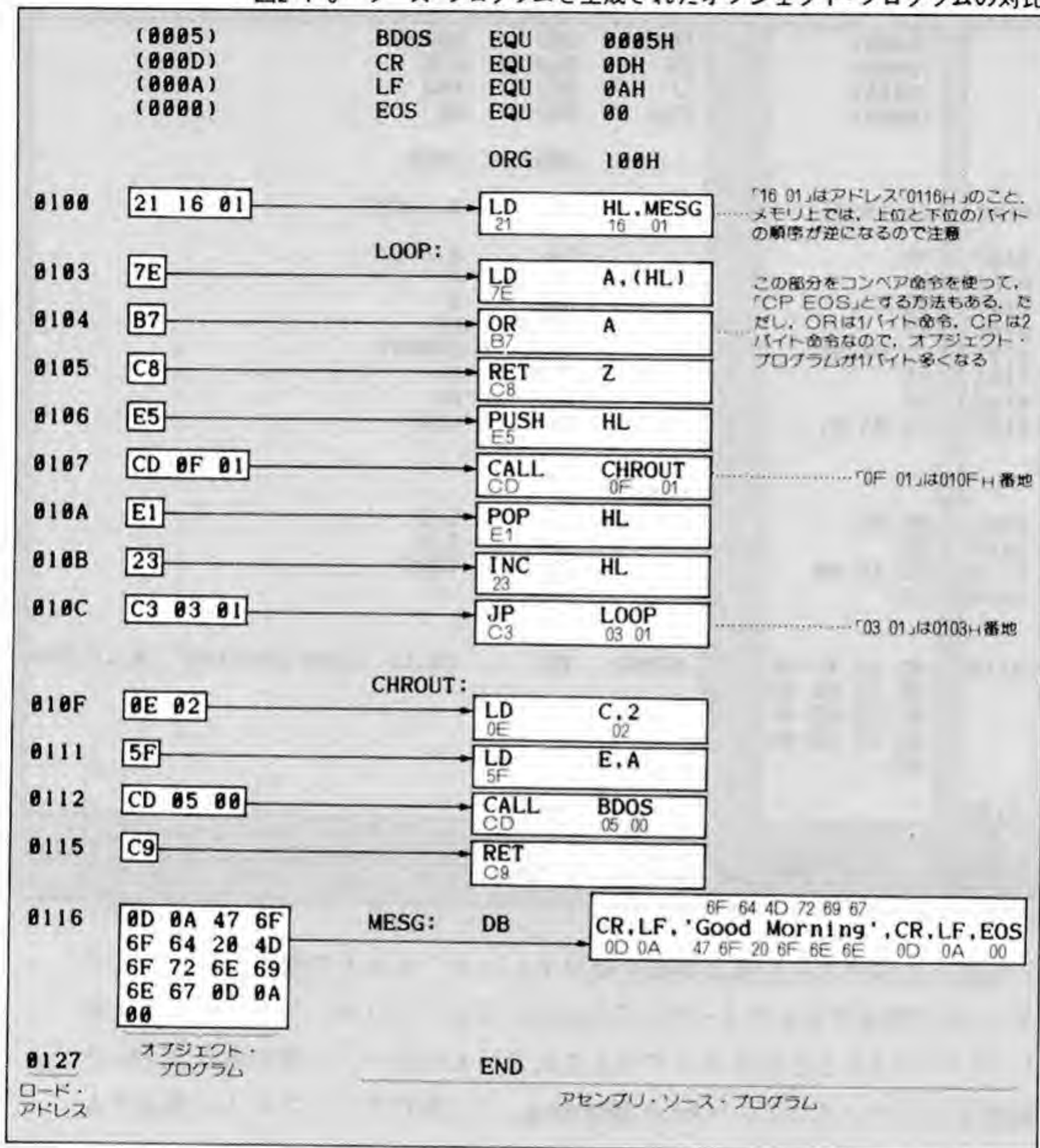
**図2-1-5 すべてのコメント類を省いたアセンブルリスト**

```
        (0005)          BDOS    EQU     0005H
        (000D)          CR      EQU     0DH
        (000A)          LF      EQU     0AH
        (0000)          EOS     EQU     00

                                ORG     100H

0100    21 16 01                LD      HL,MESG
                        LOOP:
0103    7E                      LD      A,(HL)
0104    B7                      OR      A
0105    C8                      RET     Z
0106    E5                      PUSH    HL
0107    CD 0F 01                CALL    CHROUT
010A    E1                      POP     HL
010B    23                      INC     HL
010C    C3 03 01                JP      LOOP

                        CHROUT:
010F    0E 02                   LD      C,2
0111    5F                      LD      E,A
0112    CD 05 00                CALL    BDOS
0115    C9                      RET

0116    0D 0A 47 6F     MESG:   DB      CR,LF,'Good Morning',CR,LF,EOS
        6F 64 20 4D
        6F 72 6E 69
        6E 67 0D 0A
        00

0127                            END
```

↑ロード・アドレス　↑オブジェクト・プログラム（1バイトごとにスペースを入れて区切ってある）　↑アセンブリ・ソース・プログラム

では，このリストを基に解説を続けましょう．リスト左側の，ロード・アドレス(このオブジェクト・プログラムを，メモリ上にロードする場合のメモリ・アドレス)とともにタイプアウトされているのがマシン語であり，CPUを直接コントロールして，「Good Morning」とCRTディスプレイに表示するプログラムです．このマシン語のことを，オブジェクト・プログラムと呼びます．私たちが求めている，CPUを動かす目的(Object)のプログラムという意味です．

ここで，このオブジェクト・プログラムと，その素であるソース・プログラムを，もっとよく対比する意味で，両者を重ね合わせてみましょう．ソース・プログラムの各ラインごとの命令語が，それぞれ1つ(1～3バイト構成)のマシン語に直接変換されていることがよくわかります．

図2-1-6　ソース・プログラムと生成されたオブジェクト・プログラムの対比

ここで，このプログラムのアルゴリズム(考え方や手順の流れ)を図解しておきます.* メッセージの終了を検知するために，メッセージの最後に「00」を置き，メッセージデータを頭から順に1バイトずつ取り出しては表示し，00が来れば終了する，というのがミソです．

* このプログラムは，CP/M(5章で解説)上で実行するものとして作られているが，CP/Mの内部機能を利用することで，より簡単なプログラムにすることも可能(APPENDIX 1参照)．

ここでの手法は，1文字ずつAレジスタに取り出した，表示するためのメッセージデータに対して，そのつど「OR A」命令を実行し，Zフラグが立つかどうか（Aレジスタが00かどうか）をチェックしているものです．「OR A」の代わりに，「CP EOS」命令を使ってもよいでしょう．ただし，OR命令は1バイト，CP命令は2バイトの命令です．

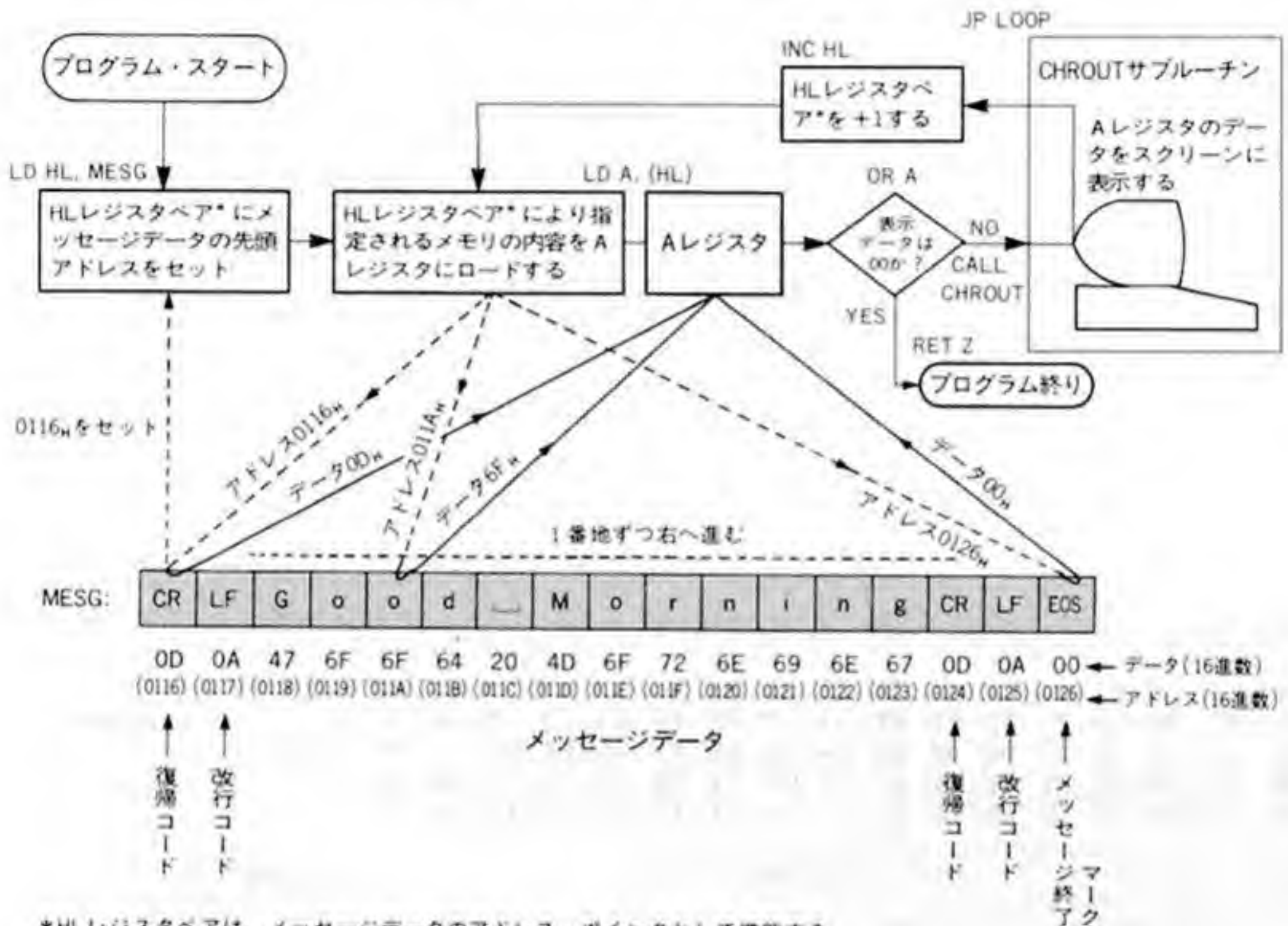



**図2-1-7 当プログラムのアルゴリズム**

では，このオブジェクト・プログラムを実際にコンピュータのメモリ上にロード（格納）して，プログラムを実行してみましょう．

アセンブルにより生成されたオブジェクト・プログラムを，どのようにしてメモリ上にもってきて，それを実行するのかについては，後ほど解説します．ここではただ，メモリ上にロードされた状態に注目してください．

ソース・プログラムの冒頭部分にある，

```
ORG 100H
```

という命令により，このソース・プログラムのオブジェクトは，メモリ・アドレスの$0100_H$を先頭にして生成されています．

ここで重要な命令語のひとつ，「ORG」の機能を覚えておきましょう．ORGは「オリジン」と読みます．

**●擬似命令　ORG**——— CPUに対する命令語ではなく，アセンブラに対してアセンブル時の処理を指示する命令語であり，このような命令を「擬似命令」と呼ぶ．アセンブラは，「ORG xxxxH」が書かれた行以降のオブジェクト・プログラムの生成を，$xxxx_H$を先頭番地として作り出していく

では，オブジェクト・プログラムをコンピュータにロードした後の，アドレス$0100_H$からのメモリ内容をダンプして見てみましょう．

図2-1-8　メモリ上にロードされたオブジェクト・プログラム

```
-D100 14F↵ ……アドレス0100H～014FHのメモリ内容をダンプする　……当プログラムのオブジェクト・プログラム

0100  21 16 01 7E B7 C8 E5 CD 0F 01 E1 23 C3 03 01 0E  !..~.......#....
0110  02 5F CD 05 00 C9 0D 0A 47 6F 6F 64 20 4D 6F 72  ._......Good Mor
0120  6E 69 6E 67 0D 0A 00 FF FF FF FF FF FF FF FF FF  ning............
0130  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ................
0140  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ................
-
アドレス              メモリ内容                           左の16進表示の各バイトに対
                                                           するアスキー表示
      *↵はキャリッジ・リターンを示す
```

オブジェクト・プログラムをロードする前のメモリには，アドレス$0100_H$付近の全体に，データ「$FF_H$」を入れておきましたので，ロード状態がよくわかると思います．アドレス$0100_H$のデータ「$21_H$」から，$0126_H$のデータ「$00_H$」までがロードされたオブジェクト・プログラムです．これを，先のアセンブルリストの左側のオブジェクト・プログラム部と比較してください．

また，アセンブルリスト上に示されているオブジェクト・プログラムは，次の図のように，1バイトずつ矢印の順にメモリ上にロードされています．*

*ただし，マイクロソフト社のアセンブラ「M80」は，アセンブルリストのオブジェクト・プログラム部の表示形式が一部異なっている．これについては図3-1-2および9.6章を参照．

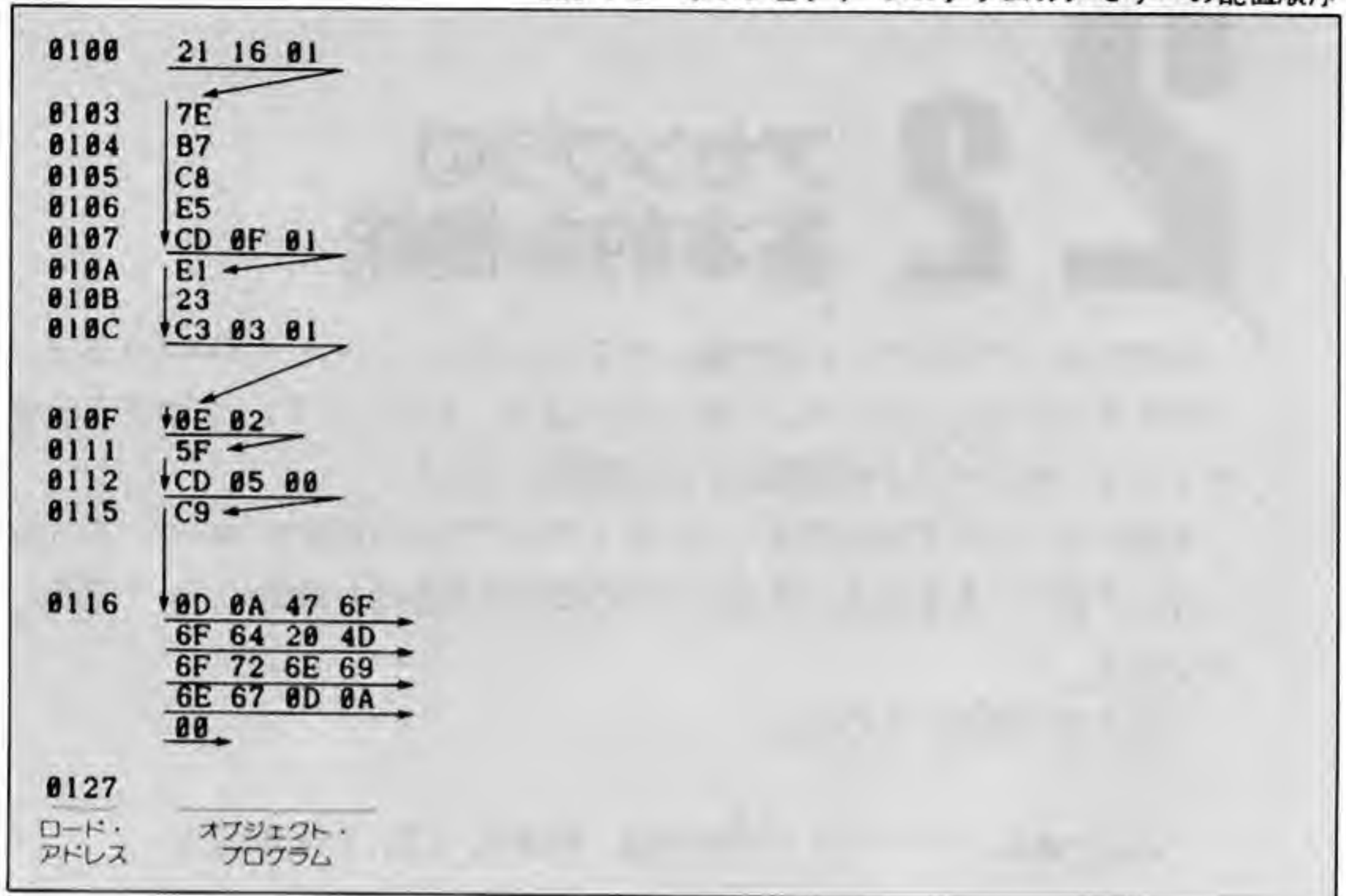
図2-1-9　オブジェクト・プログラムのメモリへの配置順序

では，このプログラムを実行してみましょう．ここでは，アドレス $0100_H$ から実行を開始した場合の，その実行結果だけを示します．

図2-1-10　オブジェクト・プログラムの実行

```
A>MSGOUT ↵ ………… 当プログラム(プログラム名MSGOUT)を実行するコマンド
 ………………………… 復帰・改行
Good Morning …… メッセージ   プログラムの実行により，メッセージが表示されている
 ………………………… 復帰・改行
A> ……………………… 実行終了(当プログラムによる表示ではない)
```

# 2/2 アセンブラの基本的な機能

前節では，アセンブリ言語で書いたアセンブリ・ソース・プログラムと，それをアセンブルして，マシン語——つまり，オブジェクト・プログラムを作りだすアセンブラとの関係をざっと解説しました．

本節では，このアセンブリ・ソース・プログラムの内容について，もう少し詳しく見ていきながら，アセンブラの働きの基本的な機能について概説しましょう．

ここで主に解説するのは，

- **シンボル** ———リストの例では，BDOS，CR，LOOP など
- **ラベル** ———リストの例では，LOOP，CHROUT，MESG
- **擬似命令** ———リストの例では，ORG，DB，END
- **CPU 命令**———リストの例では，「LD HL，MESG」，「OR A」，「RET Z」など

の 4 つの基本的項目です．

今回の解説をするためのアセンブルリストの内容は，前回のものと同じですが，今回のものはリストの各行に，ライン No.がつけられています．これはアセンブラによって付加された，ただの行番号であり，アセンブラの基本的な機能には関係ありません．

では，それぞれの事項について順に解説していきましょう．ただしここでは，あくまでそれぞれの項目の基本的な機能を解説するにすぎません．実際の使い方や，さらに詳細なことについては，次章以降で解説していきます．

## シンボル

まず，シンボルの部分をすべてピックアップしてみましょう．印をつけた部分がそれに当たります．

図2-2-1 シンボルに関する部分

```
        (0005)       0001 [BDOS]     EQU    0005H
        (000D)       0002 [CR]       EQU    0DH
        (000A)       0003 [LF]       EQU    0AH
        (0000)       0004 [EOS]      EQU    00

                     0005            ORG    100H

0100    21[1601]     0006            LD     HL,[MESG]
                     0007 [LOOP:]
0103    7E           0008            LD     A,(HL)
0104    B7           0009            OR     A
0105    C8           0010            RET    Z
0106    E5           0011            PUSH   HL
0107    CD[0F01]     0012            CALL   [CHROUT]
010A    E1           0013            POP    HL
010B    23           0014            INC    HL
010C    C3[0301]     0015            JP     [LOOP]

                     0016 [CHROUT:]
010F    0E02         0017            LD     C,2
0111    5F           0018            LD     E,A
0112    CD[0500]     0019            CALL   [BDOS]
0115    C9           0020            RET

0116    [0D][0A]476F 0021 [MESG:]    DB     [CR],[LF],'Good Morning',[CR],[LF],[EOS]
        6F64204D
        6F726E69
        6E67[0D][0A]
        [00]

0127                 0022            END
```

オブジェクト・プログラム中のシンボルに関係するデータの部分　ライン№　ソース・プログラム中のシンボルの部分

これらのシンボルの中で，「LOOP :」，「CHROUT :」，「MESG :」の3つは，シンボルの中でも特に「ラベル」と呼ばれ，これらについては，次の項で解説します．

シンボルは，特定の数値を「名前」(つまりシンボル名)で表すものです．例えば，ライン No.1，および No.19 にある「BDOS」というシンボルには，数値「$0005_H$」が定義づけられています．この BDOS に $0005_H$ を定義づけているのは，ライン No.1 の擬似命令「EQU」を使った，

```
BDOS  EQU  0005H
```

という行です．
EQU は，等号[=]の意味であり，詳しくは後ほど擬似命令の項で解説します．同様に，ライン No.2 および No.21 の「CR」というシンボルは，数値「$000D_H$」を表し，ライン No.4 および No.21 の「EOS」というシンボルは，「$0000_H$」を表します．

シンボルによって表現可能な数値の範囲は，0～$FFFF_H$，10 進数では 0～65535 です．つまり，符号なしの 16 ビット(2 バイト)で表せる範囲です．

さて，シンボルがどのように使われているかは，上のリストを見るだけでも，大体わかると思いますが，もし，シンボルを使用せずにソース・プログラムを書くとどうなるでしょう．シンボルを使用せず(ただし，ラベルは使う)，そのまま絶対値*で記述したソース・プログラムをアセンブルしたアセンブルリストを右に示します．ライン No.はつけてありませんが，**図 2－2－1** とよく比較してください．生成されているオブジェクト・プログラムは，もちろん一致しています(「DB」については，擬似命令の項で解説します)．

このように，シンボルを使わない場合は，それぞれの数値を絶対値で書かなければなりません．この例のような，小さいプログラムで，ほんの数か所しか数値を使わない場合は，シンボルを使わず直接に絶対値で書いてもさして変わりはありません．

しかし，シンボルは，それぞれの数値を表す代名詞だけではなく，その意味を私たちの言葉で表現してくれます．シンボル名には，字数などの制限がありますが，読めばその意味がわかる名前を，プログラマが自由につけることができます．ここでの例では，「CR」は Carriage Return，「EOS」は End

*ここで絶対値というのは，数学でいう絶対値ではなく，データを直接数字で書いたものを意味する．例えば，$0005_H$(オブジェクトは0500)．

図2-2-2　シンボルを使わない場合の同一プログラム

```
                              ORG     100H

0100   211601                 LD      HL,MESG
                      LOOP:
0103   7E                     LD      A,(HL)
0104   B7                     OR      A
0105   C8                     RET     Z
0106   E5                     PUSH    HL
0107   CD0F01                 CALL    CHROUT
010A   E1                     POP     HL
010B   23                     INC     HL
010C   C30301                 JP      LOOP

                      CHROUT:
010F   0E02                   LD      C,2
0111   5F                     LD      E,A
0112   CD0500                 CALL    0005H
0115   C9                     RET

0116   0D0A476F       MESG:   DB      0DH,0AH,'Good Morning',0DH,0AH,00
       6F64204D
       6F726E69
       6E670D0A
       00

0117                          END
```

シンボルを使わず，数値を直接記入している

シンボルを使わなくても，オブジェクト・プログラムは同じ

Of String という意味を表しているわけです．

シンボルを使用するもうひとつの有効性は，ソース・プログラムの変更に対するものです．例えば，ここの例では，シンボル(CR および LF)が使われるのは，ライン No.21 にそれぞれ 2 回出てくるだけです．しかし，大きなプログラムになると，同じ意味の数値が，随所に何十か所も出てくるでしょう．そのような場合，ある意味の数値を変更するとなるとたいへんです．その数値に関係する部分を，何十か所も書き直さなくてはなりません．

ところがシンボルを使っていると，実に簡単です．例えば，ここでは CR＝$0D_H$ですが，これを，$0F_H$に変更するとしましょう．ソース・プログラムに CR が何十か所，何百か所あっても，変更はただ 1 か所，EQU による宣言部の $0D_H$ を，

```
CR  EQU   0DH
          ⇧
          0FH
```

のように書き換えるだけでよいわけです.

## ラベル

ラベルもシンボルの仲間ですが，シンボルを特定の目的に使う場合にラベルと呼びます．ラベルやシンボルは，アセンブラの根幹をなす重要な部分ですが，本章ではその概要を理解しておいてください．

ではまず，ラベルの部分をすべてピックアップしてみましょう．

図2-2-3　ラベルに関する部分

```
        (0005)      0001 BDOS     EQU    0005H
        (000D)      0002 CR       EQU    0DH
        (000A)      0003 LF       EQU    0AH
        (0000)      0004 EOS      EQU    00

                    0005          ORG    100H

0100    21[1601]    0006          LD     HL,[MESG]
                    0007 [LOOP:]
[0103]  7E          0008          LD     A,(HL)
0104    B7          0009          OR     A
0105    C8          0010          RET    Z
0106    E5          0011          PUSH   HL
0107    CD[0F01]    0012          CALL   [CHROUT]
010A    E1          0013          POP    HL
010B    23          0014          INC    HL
010C    C3[0301]    0015          JP     [LOOP]

                    0016 [CHROUT:]
[010F]  0E02        0017          LD     C,2
0111    5F          0018          LD     E,A
0112    CD0500      0019          CALL   BDOS
0115    C9          0020          RET

[0116]  0D0A476F    0021 [MESG:]  DB     CR,LF,'Good Morning',CR,LF,EOS
        6F64204D
        6F726E69
        6E670D0A
        00
0127                0022          END
```

このリストから，ラベルはまずメモリ上のアドレスを表していることがわかります．ソース・プログラム中の命令行の先頭に書かれるラベル(シンボル名の後にコロン[：]をつける)は，アセンブラの実行によりソース・プログラムがオブジェクト・プログラムに変換されるとき，その命令行のオブジェクトコード(1～3バイトのマシン語)が置かれるメモリの1バイト目の番地(アドレス値)を持ちます．

例えばライン No.21の，ラベル「MESG :」の行は，擬似命令「DB」により，それに続くメッセージのデータが，オブジェクト・プログラムに組み込まれます．この場合，ラベル「MESG」は，オブジェクト・プログラムがメモリ上にロードされた場合のメッセージデータの先頭アドレスの値を持つのです．つまりアセンブルの結果，

```
MESG = 116H
```

ということになり，「MESG」というラベルは，数値(アドレス値) $0116_H$を持つことになりました．よって，このラベルを使用しているライン No.6の MESG は，このアドレス値に置き換わり，この行はアセンブラの内部では，

```
LD  HL , 116H
```

となるわけです．

ここで，アセンブラ実行前のソース・プログラムの状態のことを考えてみましょう．この命令行，

```
LD  HL , MESG
```

は，CRT ディスプレイに表示するためのメッセージデータが置かれたメモリの先頭アドレス値を，アドレス・ポインタとして使用する HL レジスタペアにロードすることが目的です．

しかし，メッセージが置かれているのはプログラムの後部ですので，その部分までアセンブラが実行され，一通りオブジェクトコードを作り出してみないことにはそのアドレス値は判明しません．つまり，ソース・プログラムの段階ではアドレスは不明であるわけです．このことが，「2パス・アセンブラ」などといわれているものの原理です．このことは，ハンド・アセンブルを行った経験のある人にはすぐ理解できることと思いますので，簡単に説明しておきましょう．

パス1（1回目の処理）では，CPU命令自身はオブジェクトコードに変換できますが，現在アセンブルしている行より後方にあるラベルのアドレス値はまだ不明です．そこで不明のラベルを使用しているJP命令やCALL命令などのアドレス値には，とりあえず0000の2バイトを代入しながら，ソース・プログラムの最後までアセンブルして，一応パス1の処理を終わります．

この時点ではすでにラベルのアドレス値が判明しているので，パス1で代入した0000を正しいアドレス値と入れ換えます．これがパス2（2回目の処理）の仕事であり，以上で完全なオブジェクト・プログラムが生成できたわけです．しかしこれらのパスの処理は，アセンブラが自動的に行ってくれるので，このパスを特に意識する必要はありません．

ハンド・アセンブルの経験がない方は，後ほどぜひこのソース・プログラムを紙とエンピツでハンド・アセンブルしてみてください．その経験が，本章の理解に非常に役立つことでしょう．

さて本論に戻りますが，ライン No.15 の JP 命令の行も同様に考えます．このJP命令は，ライン No.8 の，「LD A ,(HL)」の命令行にジャンプするのが目的です．よって，「LD A ,(HL)」の行の先頭に，ラベル「LOOP :」を置き，JP命令はそのラベルを使って，「JP LOOP」とすればよいわけです．アセンブル後は，リストのオブジェクト・プログラムに見られるように，シンボル「LOOP」が絶対アドレス値に置き換えられています．

ライン No.12 のサブルーチンコールの行，「CHROUT」も同様です．ライン No.17～20 のプログラムが，CRTディスプレイに1文字出力するサブルーチンで，その先頭に，ラベル「CHROUT :」を置いています．よって，このサブルーチンを使用するときには，このラベル「CHROUT」を呼べばよいわけです．

このように，ラベルを使うことにより，プログラマーはソース・プログラムを書く段階で，JP 命令やサブルーチンコール命令などの，アドレス値が必要な命令を，その絶対値を意識することなく，自由にプログラミングできることになります．

## 擬似命令

まず，擬似命令の部分をすべてピックアップしてみましょう．

図2-2-4　擬似命令に関する部分

```
        (0005)      0001 BDOS     EQU    0005H
        (000D)      0002 CR       EQU    0DH
        (000A)      0003 LF       EQU    0AH
        (0000)      0004 EOS      EQU    00

                    0005          ORG    100H

0100    211601      0006          LD     HL,MESG
                    0007 LOOP:
0103    7E          0008          LD     A,(HL)
0104    B7          0009          OR     A
0105    C8          0010          RET    Z
0106    E5          0011          PUSH   HL
0107    CD0F01      0012          CALL   CHROUT
010A    E1          0013          POP    HL
010B    23          0014          INC    HL
010C    C30301      0015          JP     LOOP


                    0016 CHROUT:
010F    0E02        0017          LD     C,2
0111    5F          0018          LD     E,A
0112    CD0500      0019          CALL   BDOS
0115    C9          0020          RET

0116    0D0A476F    0021 MESG:    DB     CR,LF,'Good Morning',CR,LF,EOS
        6F64204D
        6F726E69
        6E670D0A
        00

0127                0022          END
ロード・ オブジェクト・ ラインNo      アセンブリ・ソース・プログラム
アドレス プログラム
```

擬似命令は次項のCPU命令と根本的に異なる命令です．

Z-80，8080，8085などのCPUには，どれでも必ずそれぞれのインストラクション表(CPU命令の一覧表)が用意されています．みなさんの手もとに，このインストラクション表があれば，ちょっと目を通してください．この表は，マシン語やアセンブラの参考書の巻末にもよく載っています．

ところが，これらの表には上のリストに示したEQU，ORG，DBなどの命令は，いくら捜しても見つかりません．なぜならこれらは，CPUをコントロールする命令ではないからです．

**＊重要！**

『擬似命令は，CPUをコントロールする命令ではなく，アセンブラの処理をコントロールする命令である．よって，擬似命令自身に対するマシン語は生成されない』

つまり，擬似命令とは，アセンブラ自身の動作のしかたに対して指示を出す命令です．そこで，擬似命令は，「アセンブラ制御命令」とも呼ばれます．擬似命令とCPU命令との違いを理解することは，アセンブラ全体を理解するための非常に重要なポイントです．ここの所は明確にしておきましょう．

では，リストに登場する擬似命令の基本的なことを，順に解説していきましょう．*

＊擬似命令についての詳細は，8.4章で再度解説する．

## ＊ EQU

EQU 擬似命令は，EQUate，つまりイコール[＝]の意味です．EQU を[＝]記号に置き換えて考えれば，あまり説明する必要もなさそうです．例えば，

```
BDOS  EQU  0005H
```

の命令行は，「シンボル BDOS を，この行以降では，数値 $0005_H$ と等しいと考えよ」ということを，アセンブラに指示する命令です．数値はいずれの場合も 2 バイトの値であり，

```
CR    EQU  0DH
```

の命令行は，シンボル「CR」を $000D_H$ として定義します．

EQU 擬似命令によって定義され，数値を与えられたシンボルは，それを定義した行以降でのみ有効です．次に示すように，ソース・プログラムの中で，それを定義した行以前での使用は，エラーとなります．

（シンボル ABCD は定義されておらず，その使用はできない）
↑
```
ABCD EQU 1234H
```
↓
（シンボル ABCD は，数値 $1234_H$ として，使用できる）

## ＊ ORG

ORG 擬似命令は，ORiGin（初め，発端）の意味です．前節でも解説しましたが，例えば命令行，

```
ORG  1234H
```

は，この行以降のアセンブラの実行によるマシンコードの生成を，アドレス $1234_H$ を先頭として順に行うように，アセンブラに指示する命令です．

本章で例題にしているソース・プログラムは，CP/M*上で実行するプログラムのため，$0100_H$をスタート・アドレスとしていますが，これを$9000_H$のスタート・アドレスとなるように，再アセンブルしてみましょう．ソース・プログラムの「ORG 100H」を，「ORG 9000H」に変更すればよいわけです．その結果のアセンブルリストを示します．

図2-2-5　オリジンを変更して再アセンブルした結果のリスト

```
        (0005)       0001 BDOS    EQU    0005H
        (000D)       0002 CR      EQU    0DH
        (000A)       0003 LF      EQU    0AH
        (0000)       0004 EOS     EQU    00

                     0005         ORG    9000H

9000    211690*      0006         LD     HL,MESG
                     0007 LOOP:
9003    7E           0008         LD     A,(HL)
9004    B7           0009         OR     A
9005    C8           0010         RET    Z
9006    E5           0011         PUSH   HL
9007    CD0F90*      0012         CALL   CHROUT
900A    E1           0013         POP    HL
900B    23           0014         INC    HL
900C    C30390*      0015         JP     LOOP

                     0016 CHROUT:
900F    0E02         0017         LD     C,2
9011    5F           0018         LD     E,A
9012    CD0500       0019         CALL   BDOS
9015    C9           0020         RET

9016    0D0A476F     0021 MESG:   DB     CR,LF,'Good Morning',CR,LF,EOS
        6F64204D
        6F726E69
        6E670D0A
        00

9027                 0022         END
```

*印の3つのアドレス値が，0100H スタートの場合と異なっていることに注目

ロード・アドレス　オブジェクト・プログラム　ラインNo.　アセンブリ・ソース・プログラム

このように，オブジェクト・プログラムの先頭アドレスが$9000_H$になり，そこから順にマシンコードが生成されています．リスト左端のアドレス部と，特にCALL命令やJP命令の飛び先アドレスなどに注目して，以前のリスト**図2-2-4**などと比較してください．

*詳しくは5章を参照．

## * DB

DB擬似命令は，Define data Byte(データの1バイトずつの定義)の意味であり，DBの後に指定する各種の形式のデータをこのDB擬似命令が存在するメモリ・アドレスから1バイト単位でオブジェクト・プログラム中に組み込む(確保する)ことをアセンブラに指示します．どのような形式のデータが記述できるのか，それらがどうオブジェクト・プログラム中に組み込まれるのかを，**図2－2－4**のリストを参考に解説しましょう．

```
DB   CR,LF,~,CR,LF,EOS
```
(ソース・プログラムの記述)

```
---   0D 0A ~ 0D 0A 00
```
(オブジェクト・プログラムに組み込まれた状態，16進)

上の形式は，DB擬似命令の後にシンボルを指定した例で，それぞれのシンボル(CR,LF,EOS)が持つ値が，1バイト単位で確保されます．複数個のシンボルを同一行に指定する場合は，このように，カンマ[ , ]で区切ります．

これらのシンボルは，プログラムの冒頭のEQU擬似命令で，CR＝$0D_H$，LF＝$0A_H$，EOS＝$00_H$に定義されていますので，オブジェクト・プログラム中には上に示したように，それらの値が組み込まれます．

シンボルはいずれの場合も，2バイトの数値として定義されていますが，1バイトを単位とするDB擬似命令で取り扱うには，上位のバイトは「00」でなければなりません.*　上記のシンボルは，例えばCR＝$0D_H$の場合は$000D_H$というように，上位バイトが「00」であるために，DB擬似命令で使用することが可能なのです．

*アセンブラによっては，上位のバイトが「FF」でもよいものがある．

次の例は，文字列の指定です．

```
DB 'Good Morning'
```

（ソース・プログラムの記述）

```
~ 47 6F 6F 64 20 4D 6F 72 6E 69 6E 67 ~
```

（オブジェクト・プログラムに組み込まれた状態，16 進）

上の形式は，ストリング（文字列）を指定した例で，クォート[']で囲った文字列の 1 文字ごとのアスキーコードが確保されます（APPENDIX 4のアスキーコード表を参照）．複数個の文字列をシンボルなどを混在させ，同一行に指定する場合は，

```
DB '文字列1','文字列2',CR,LF,…
```

のようにカンマ[,]で区切ります．

「Good Morning」に対するアスキーコードは，G=$47_H$，o=$6F_H$，d=$64_H$，…なので，オブジェクト・プログラムには，上に示したように組み込まれます．

次の例は，数値の直接指定です．

```
DB 0DH,0AH,00
```

（ソース・プログラムの記述）

```
~ 0D 0A 00 ~
```

（オブジェクト・プログラムに組み込まれた状態，16 進）

上の形式は1バイトの数値を絶対値で指定する例で，そのままの数値が確保されます．ただし，数値は1バイト（符号なしの8ビット）で表せる数（0～$FF_H$，10進数では0～255）の範囲でなければなりません．

同一行に複数個を指定する場合には，例によってカンマ[,]で区切ります．これらの具体例は，**図2-2-2**「シンボルを使わない場合の同一プログラム」のリストを参照してください．なお，本章のソース・プログラムでも示されているように，シンボル，文字列，数値などを1つのDB擬似命令の行に混在させることが可能です．

その他に，この種類の擬似命令には，2バイト単位*でデータを指定するDW（Define data Word）や，データの確保ではなく，メモリエリアを指定したバイト分だけ確保するDS（Define Storage）などがあります．

### * END

END擬似命令は，文字どおりのEND（終り）の意味です．END擬似命令は，ソース・プログラムの終了をアセンブラに知らせるものであり，アセンブラは，END擬似命令以降の行を，何が書かれていようと無視します．

なおこのEND擬似命令は，アセンブラの種類やその使い方によっては，書かなくてもよい場合もあります．

その他にもよく使われる擬似命令としては，条件つきアセンブルを可能にする，IF，ENDIF擬似命令などがありますが，これらについては8.4章で解説します．

＊これをワード単位と呼ぶ．Z-80アセンブラでは2バイトをワードと呼ぶが，どのコンピュータでも2バイトのことをワードと呼ぶわけではないので注意が必要．

## CPU命令

まず，CPU 命令に関する部分をすべてピックアップしてみましょう．

図2-2-6　CPU命令に関する部分

```
        (0005)           0001 BDOS     EQU    0005H
        (000D)           0002 CR       EQU    0DH
        (000A)           0003 LF       EQU    0AH
        (0000)           0004 EOS      EQU    00

                         0005          ORG    100H

0100   [211601]   |      0006         [LD     HL,MESG]
                  |      0007 LOOP:
0103   [7E    ]   |      0008         [LD     A,(HL)
0104   [B7    ]   |      0009          OR     A
0105   [C8    ]   C      0010          RET    Z
0106   [E5    ]   P      0011          PUSH   HL
0107   [CD0F01]   U      0012          CALL   CHROUT
010A   [E1    ]   命     0013          POP    HL
010B   [23    ]   令     0014          INC    HL
010C   [C30301]   に     0015          JP     LOOP]
                  よ
                  る     0016 CHROUT:
010F   [0E02  ]   オ     0017         [LD     C,2
0111   [5F    ]   ブ     0018          LD     E,A
0112   [CD0500]   ジ     0019          CALL   BDOS
0115   [C9    ]   ェ     0020          RET]
                  クトコード

0116    0D0A476F  デ     0021 MESG:    DB     CR,LF,'Good Morning',CR,LF,EOS
        6F64204D  ー
        6F726E69  タ
        6E670D0A  コード
        00

0127                     0022          END
```

これらは『はじめて読むマシン語』でもその基礎的なことを解説しましたので，すでにご承知のことと思いますが，ニーモニックによる CPU に対する命令(インストラクション)です．

アセンブラの実行により，これらの各命令は，命令の種類によって，1 ～ 3 バイトのマシンコードに変換されます．上のリストからも，1 バイト命令，2 バイト命令，3 バイト命令のそれぞれの種類を見ることができます．

さて本章では，アセンブラによるプログラミングの全体的な姿を眺めてみました．アセンブラの大まかな姿は大体知ることができたと思います．

# 3 ソース・プログラムの基本的な書式

本章ではアセンブラのソース・プログラムを書く場合に最低限知っておかなければならない基本的な制約事項や，書式についての解説を行います．ここで解説するのは，プログラムの中身，つまりプログラミングについてではなく，プログラムを書くための大前提である，アセンブルが可能で，アセンブル・エラーのない，読みやすいソース・プログラムをいかに書くかについてです．

ソース・プログラムを作成する――つまり書くためには，プログラムの「文章」を書くための道具(ソフトウェアツール)である「エディタ」が必要です．学習用などの簡易版アセンブラの多くはマシンに組込みのBASICのエディタを利用していますが，通常のアセンブラにはエディタ専用のソフトウェアが必要です．

一般的なエディタとしては，CP／Mに標準装備の「ED」などが知られていますが，さらに使いやすいスクリーンエディタも，数社から発売されています．またワードプロセッサには，エディタの代わりに使用できるものもあります．

本章ではまず，読みやすく美しく整理されたソース・プログラムの書き方について学んでいきましょう．エディタに関しては，5章「ソフトウェア開発ツールとその機能」で解説していますので参照してください．

# 3/1 ステートメントを構成する「フィールド」と約束事

アセンブラは，アセンブリ言語で書かれたソース・プログラムを，ニーモニックの単位でマシン語に変換していきます．ソース・プログラムの各行は，CPU命令やラベル，あるいは擬似命令などで構成され，それらの基本単位は「ステートメント」と呼ばれています．そのステートメントの書き方にはある程度の規則があり，ソース・プログラムはその書式に従って記述しなければなりません．

## ステートメントの書式

アセンブリ・ソース・プログラムのそれぞれのステートメントの基本構成は，大きく分けて次に示す順に3つの部分から成り立っています．

- **シンボル部**
- **命令部**
- **コメント部**

これらの部分は次のように，それぞれ「‥‥フィールド」と呼びますが，命令部だけは，2つのフィールドで構成されています．

- **シンボル部**———シンボル・フィールド
- **命令部**————オペレーション・フィールド，および<br>アーギュメント・フィールド
- **コメント部**———コメント・フィールド

2．2章で，シンボル，ラベル，擬似命令，CPU 命令のそれぞれについて解説しましたが，この内のシンボルとラベルは「シンボル・フィールド」に，擬似命令と CPU 命令は，命令部に記述します．コメントは，［；］を前置きして，どの位置に書いてもかまいません．これらを整理して，次の表に示します．

図3-1-1　ステートメント（行）を構成する各フィールド

## 約束事

次に，ソース・プログラムを書く場合に最も基本となる各ステートメントの書式に関する「約束事」を示します．約束事は，それぞれのアセンブラにより少しずつ異なりますが，ここでは標準的なものを示します．

**(a)　1行には1つのステートメントを書く**

「1行」とは，2.1章でも解説しましたが，書始めからリターンキーの入力までをいいます．CRTディスプレイやプリンタなどの1行の字数を越えた場合の表示上の折返しのことではありません．

次の例に示すようなマルチステートメント(1行に複数のステートメントを書くこと)は誤りです．

誤り
```
ORG  100H  :  LD  HL , MESG ↵
```
*

**(b)　各ステートメントは，アーギュメント・フィールド以外のフィールドで始まり，必ずキャリジ・リターン(リターンキーの入力)で終わること**

次の例①～④に示す各ステートメントは，いずれも正しい書式です．

```
例①  LOOP :    LD      A ,(HL)     ;(コメント)↵
例②  LOOP : ↵
例③            LD      A ,(HL)     ;(コメント)↵
例④                                ;(コメント)↵
```

**(c)　各フィールドは，図3-1-1に示した順序に並べること**

次に示す例は誤りです．

誤り
```
LD        A ,(HL)    LOOP:        ;(コメント)↵
```

* CP／Mに標準装備の8080アセンブラなどではマルチステートメントが可能なものもある．その場合はBASICの場合の[ : ]ではなく，[ ! ]で区切る．

**(d) 必要のないフィールドは無視できる**

次の例①～③に示す各ステートメントは，いずれも正しい書式です．例③は，ただ1行を空けるだけのものです．

例① `LD A,(HL) ;(コメント)↲`
例② `;(コメント)↲`
例③ `↲`

**(e) [;]以降，そのステートメントの最終まで(↲まで)何が書かれていようとアセンブラはそれを無視する**

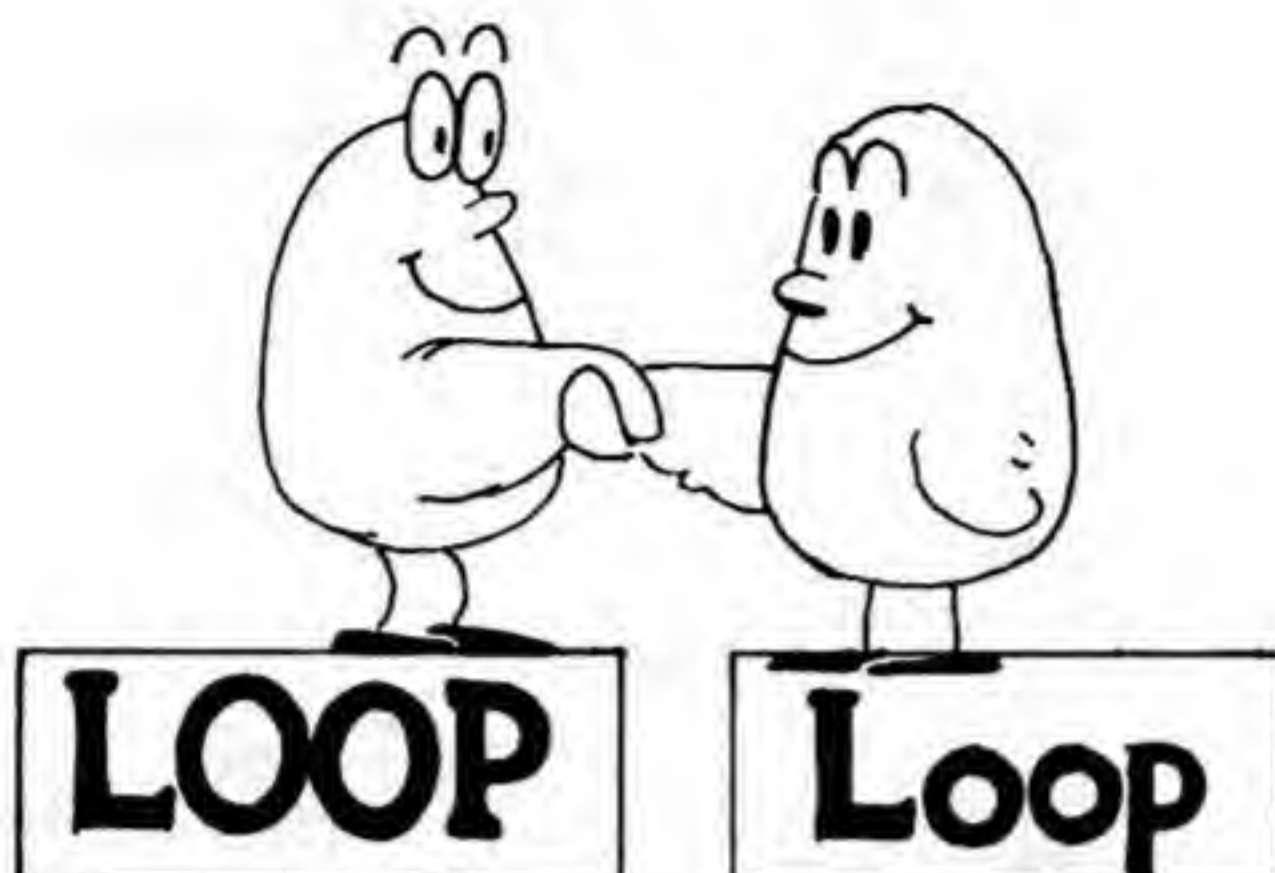
次の例①に示すステートメントのラベルやCPU命令は，アセンブルされません．

例① `;LOOP: LD A,(HL) ;(コメント)↲`

**(f) 各フィールドは1つ以上のスペースまたはタブを置いて区切ること**

各フィールドの間に1つ以上のスペースを置くことで，フィールド間の区切りとします．ただしラベルの後に[:]を置いた場合はその必要はなく，またコメント・フィールドの前には必ず[;]を置くので，この場合もスペースやタブで区切る必要はありません．

次の例①～③に示す各ステートメントは，いずれも正しい書式です．例②に見られるように，ラベルのコロン[:]およびコメントの[;]はそれ自身でフィールドを区切る意味があるのです．

例① `BDOS EQU 0005H;(コメント)↲`
例② `LOOP:LD A,(HL);(コメント)↲`
例③ `LD A,(HL)↲`

**(g)　ラベルを含めシンボルに使用できる文字やその字数には制約がある**

一般的に次の制約があります．

- 使用可能文字は，A～Z，0～9の英数字．英小文字は使用できるが，大文字との識別はされず，アセンブラの実行より，自動的に大文字に変換される．さらに数種の記号――クエスチョンマーク［？］やアットマーク［@］などが使えるアセンブラもある
- 字数は16文字までが有効*
- 1文字目は数字を使ってはいけない

例①　　LOOPとLoopは同一シンボル

例②　　誤り　　12ABCはシンボルとして誤り（1文字目が数字）

以上がソース・プログラムを書く場合の基本的な規則です．これらの約束事を守っていれば，極端な例では先のソース・プログラムを図**3-1-2**に示すように書いてもよいわけです．

このように書いたものでも，先に挙げた約束事を満足している限り，アセンブラはこれを正しいソース・プログラムとしてアセンブルします．アセンブル・エラーはなく，以前のものとまったく同じオブジェクト・プログラムが作られていることを確認してください．

*アセンブラによりもっと少ないものもある．詳しくは各アセンブラのマニュアルを参照．

図3-1-2　約束事を考慮しただけのソース・プログラム

```
0005                    BDOS                        EQU 0005H
000D                    CR EQU          0DH
000A                    LF      EQU     0AH
0000                    EOS EQU 00

                        ORG                                   100H

0100'   21 0116'*                             LD    HL,MESG
0103'                   LOOP:
0103'   7E              LD A,(HL)
0104'   B7                          OR A
0105'   C8              RET                       Z
0106'   E5                                          PUSH      HL
0107'   CD 010F'*                                   CALL      CHROUT
010A'   E1              POP     HL
010B'   23                      INC     HL
010C'   C3 0103'*               JP      LOOP

010F'                   CHROUT:
010F'   0E 02                   LD      C,2
0111'   5F                      LD      E,A
0112'   CD 0005 *               CALL    BDOS
0115'   C9                      RET

0116'   0D 0A 47 6F     MESG:   DB      CR,LF,'Good Morning',CR,LF,EOS
011A'   6F 64 20 4D
011E'   6F 72 6E 69
0122'   6E 67 0D 0A
0126'   00
ロード・  オブジェクト・
アドレス  プログラム            END
```

各フィールドをそろえずに書いたアセンブリ・ソース・プログラム

しかしこのようにゴチャゴチャしたリストでは，アセンブルは可能でもプログラムの内容を読むことが非常に困難です．やはりソース・プログラムは，読みやすいように各フィールドを縦にそろえて書く必要があります．

＊このアセンブル例では，マイクロソフト社の「M80」(リロケータブル・マクロアセンブラ，5.1章参照)を使用している．このアセンブラでは，オブジェクト・プログラム中のアドレスや2バイトの部分(〰の部分)が，メモリ上にロードされる順序とは逆――つまり，「読む順序」になっていることに注意．

# 3/2 読みやすいソース プログラムの書き方

本節では読みやすいソース・プログラムの書き方について解説しましょう．まず，例題としているソース・プログラムを次のように極端に書き，アセンブルした例を示しましょう．

図3-2-1　読みやすさをまったく考慮しないソース・プログラム

```
0005                        BDOS EQU 0005H             *図3-1-2の注釈参照
000D                        CR EQU 0DH
000A                        LF EQU 0AH
0000                        EOS EQU 00
                            ORG 100H
0100'   21 0116'*           LD HL,MESG
0103'   7E                  LOOP: LD A,(HL)
0104'   B7                  OR A
0105'   C8                  RET Z
0106'   E5                  PUSH HL
0107'   CD 010F'*           CALL CHROUT
010A'   E1                  POP HL
010B'   23                  INC HL
010C'   C3 0103'            JP LOOP
010F'   0E 02               CHROUT: LD C,2
0111'   5F                  LD E,A
0112'   CD 0005 *           CALL BDOS
0115'   C9                  RET
0116'   0D 0A 47 6F         MESG:DB CR,LF,'Good Morning',CR,LF,EOS
011A'   6F 64 20 4D
011E'   6F 72 6E 69
0122'   6E 67 0D 0A
0126'   00
                            END
ロード・  オブジェクト・        読みやすさをまったく考慮しないアセンブリ・ソース・プログラム
アドレス  プログラム
```

この例は，ソース・プログラムのスペースやコメント類を一切省き，アセンブラにとって必要な中身だけに凝縮したものですが，誤った記述ではありません．従ってアセンブルは可能であり，今までのものとまったく同じオブジェクト・プログラムが生成されています．

しかしこのソース・プログラムでは，プログラムを「読む」ことが非常に困難であり，何が何だかわかりません．プログラマーにとって，目的どおりの正しい動作をするプログラムを書くことが第一条件ですが，それと同じくらい読みやすいソース・プログラムを書くことが重要なのです．

## タブキーを使って各フィールドを縦にそろえる

読みやすく，わかりやすいソース・プログラムを書く基本の第1は，前節でも述べたように各フィールドを縦にそろえることです．

エディタでソース・プログラムを書く場合，各フィールドの先頭を縦にそろえるときに使用するのがタブキーです．ここで，タブの機能について解説しておきましょう．ほとんどのパーソナル・コンピュータには，キーボードの左端に TAB キーがついていますので，みなさんもその使い方はご存じのことと思いますが，いちおう次の実験をしてみましょう．

エディタを起動し，その入力モードの状態か，あるいはBASICが起動している状態で次のようにキー入力してみましょう．TAB は，TABキーの入力を示します．

1 TAB 12 TAB 123 TAB 1234 TAB 12345 TAB 123456 TAB 1234567 ↵
1234567 TAB 123456 TAB 12345 TAB 1234 TAB 123 TAB 12 TAB 1 ↵

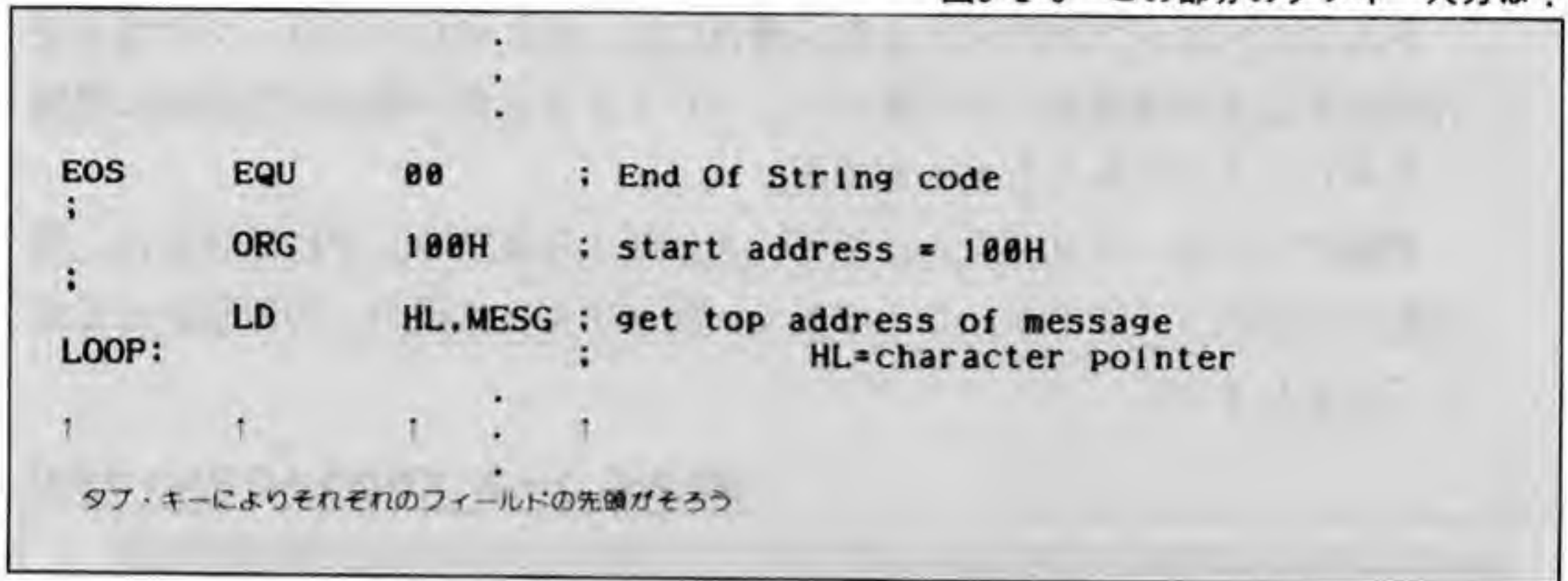
一般的なエディタのタブは8文字おきに設定されています．つまり，上のように入力しても，CRTディスプレイやプリンタのリスト上の表示には，各フィールドの先頭が，8文字おきに縦にそろうわけです．その表示例を次に示します．

図3-2-2　タブ機能の実例

```
1       12      123     1234    12345   123456  1234567
1234567 123456  12345   1234    123     12      1
↑       ↑       ↑       ↑       ↑       ↑       ↑
        タブキーを使うと8文字ごとに文字列の先頭がそろう
```

もう一例，ソース・プログラムでの具体例を示しましょう．**図3－1－1**のリストの次の部分の入力方法です．

**図3-2-3　この部分のタブキー入力は？**

```
                     .
                     .
                     .
EOS     EQU     00      ; End Of String code
;
        ORG     100H    ; start address = 100H
;
        LD      HL,MESG ; get top address of message
LOOP:                   ;          HL=character pointer
                     .
↑       ↑       ↑    .  ↑
                     .
  タブ・キーによりそれぞれのフィールドの先頭がそろう
```

この部分は，次のようにキー入力すればよいわけです．

EOS [TAB] EQU [TAB] 00 [TAB] ；(コメント)↵
；↵
[TAB] ORG [TAB] 100 H [TAB] ；(コメント)↵
；↵
[TAB] LD [TAB] HL,MESG [TAB] ；(コメント)↵
LOOP: [TAB] [TAB] [TAB] ；(コメント)↵

このようにキー入力することにより，タブキーを入力した部分( [TAB] )が自動的に必要な数だけのスペースに展開され，CRTディスプレイ上には上のリストのように，各フィールドの先頭がそろって表示されます．

タブキーは，いくつ続けて入力してもかまいません．目的のフィールドの頭の位置にカーソルがくるまで入力すればよいのです．

## ブロックごとに行を空ける

読みやすいソース・プログラムを書く基本の第2は，プログラミング作法に関係する問題です．つまり，論理的にわかりやすく構造的なプログラムを

作ることに関係します．プログラミング作法の基本については7章で解説しますが，その最も基本的なことは，プログラム全体を構造的にいくつかのブロック（モジュールともいう）に区切ることです．

さらにソース・プログラムを書く場合には，それぞれのブロックの境を空行や記号文字の連続などで分離して，プログラム全体の構造を視覚的に把握しやすいようにすることが大切です．

例題のソース・プログラムはこのことに従って各ブロックに区分され，構造的に示されていますが，ここでもう一度「ブロック区分」の観点から見直してみましょう．

図3-2-4　ソース・プログラムのブロック分け

```
;----------------------------;
;     MESSAGE OUT PROGRAM    ;
;----------------------------;
;
BDOS    EQU     0005H           ; system call entry point
CR      EQU     0DH             ; Carri
LF      EQU     0AH             ; Line
EOS     EQU     00              ; End O
;
        ORG     100H            ; start a                0H
;
        LD      HL,MESG         ; get top address of message
LOOP:                           ;          HL=character pointer
        LD      A,(HL)          ; get character code to output
        OR      A               ; end o
        RET     Z               ; if 'Z                 s program
        PUSH    HL              ; save                  er
        CALL    CHROUT          ; chara
        POP     HL              ; restore character pointer
        INC     HL              ; pointer goes up
        JP      LOOP            ; jump for next character
;
;------ 1 character out subroutine ------
;
CHROUT:
        LD      C,2             ; CP/M
        LD      E,A             ;
        CALL    BDOS            ;                       ction
        RET
;
;------ string data area for message ------
;
MESG:   DB      CR,LF,'Good Morning',CR,L
;
        END                     ; list end
```

このソース・プログラムの構造は，上から順に，

- **表題部**
- **シンボル定義部**
- **オリジン設定部**
- **メインルーチン部**
- **サブルーチン部**
- **文字列エリア部**

の各ブロックから構成されています．

これらのソース・プログラムのブロックの構成が空行によって区分され，プログラムの構成が「読み」やすくなっていることがわかります．

空行を挿入するには，行の先頭に[；]を置いて↲を入力します．また上のリストのように，ラベルを使うことも効果的です．

```
LOOP：↲
        LD      A,(HL)    ；(コメント)↲
```

このように，ラベル「LOOP：」の後は何も入力せずに↲を入力します．そうするとラベルの後にスペースができ，ラベルの箇所で何らかの境を表すことができます．ただしソース・プログラム上では，このように2つのステートメントに分かれていても，あくまで「見た目」のことであり，アセンブラから見れば，1つのステートメント，

```
LOOP：  LD      A,(HL)    ；(コメント)↲
```

と同じです．

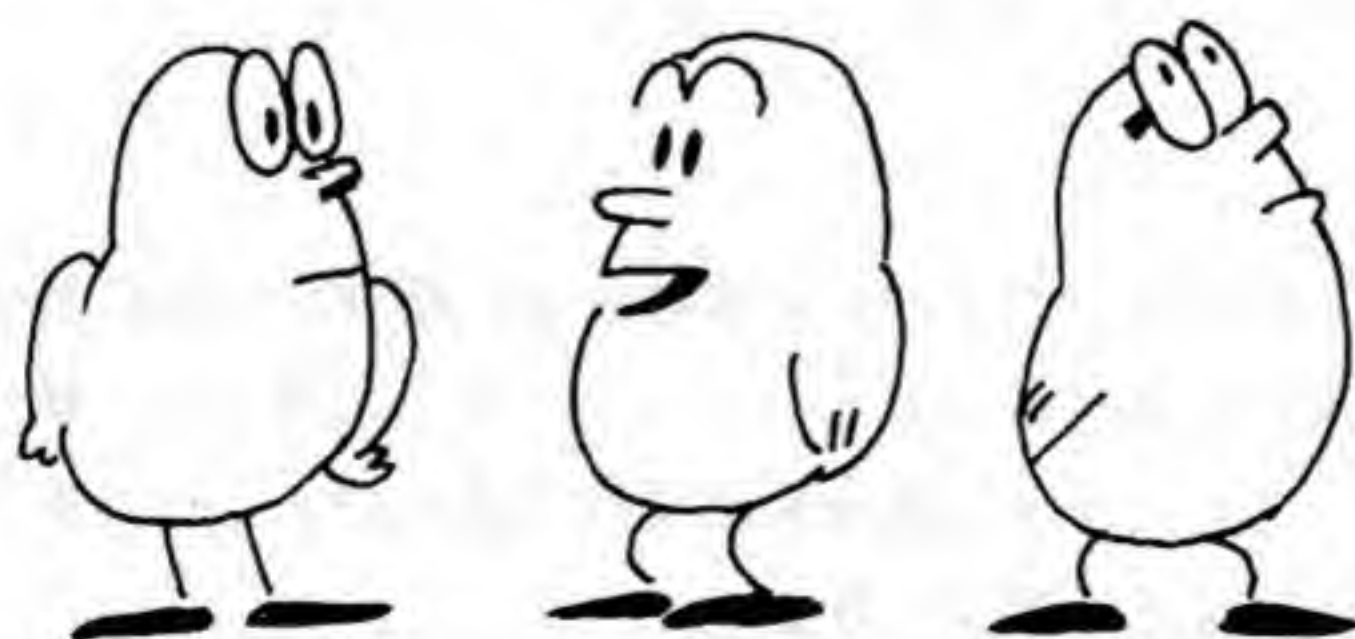
このラベルだけの行は，ラベルの文字数が多い場合には特に有効です．ラベルの文字数が多いとオペレーション・フィールドに侵入し，続けて書くとオペレーション・フィールド以降が右にずれて読みにくくなります．この問題は，2行に分けることで解決できます．

## コメントを効果的に利用する

読みやすいソース・プログラムを書く基本の第3は，コメントの効果的な利用です．図**3－2－4**のソース・プログラムにもある程度書かれていますが，コメントは，極力ていねいに，簡潔に，「人間の言葉」でプログラムを説明しておきましょう．このソース・プログラムの表題やサブルーチンの説明書きの部分などは，必要であれば1行全部，あるいは複数行を全部コメントにしてもかまいません．このような細かい部分はプログラマーのセンスの問題であり，自由にレイアウトすればよいわけです．

また，アセンブラによってはコメント部にカタカナや漢字が使用できるものもありますので，それらは英文で書かなくてもかまいません．ただしカタカナや漢字を使ったソース・プログラムを別のアセンブラでアセンブルする場合には注意が必要です．例えばCP／Mの8080用アセンブラは，カタカナ／漢字の使用が可能ですがM80は不可能です．詳しくはそれぞれのマニュアルを参照してください．

## 対象をうまく表現できるシンボルやラベルの名を用いる

最後は，あまり説明の必要もありませんがシンボルとラベルの名前のつけ方についてのことです．これらに対する制約は，それぞれのアセンブラによって多少異なります．これについては，前節3.1章の「約束事」で解説しましたが，この制限の中で，対象をうまく言い表せる名前をつければよいわけです．

# 実行速度を気にしない

さて以上説明してきた4つの事項，

- タブキーを使って各フィールドを縦にそろえること
- ブロックごとに行を空けること
- コメントを効果的に利用すること
- 対象をうまく表現できるシンボルやラベルの名を用いること

が，読みやすいソース・プログラムを書くための基本です．

アセンブラは，パーソナル・コンピュータに装備されているBASICインタープリタと異なり，マシン語そのもののプログラムを作り出します．BASICの場合には，ソース・プログラムにスペースを入れたり，リマーク(コメント)を入れたり，文字数の多い変数名を使ったりすると，それだけ実行速度が遅くなるというような弊害がありました．しかし，アセンブラの場合にはこのような問題はまったくありません．そういうことは一切気にせず，スペースやコメントを自由に使って，読みやすいソース・プログラムを書くように心がけましょう．BASICで育ったプログラマーにはこの習慣がない人が多いので，特に気をつけなければなりません．

読みやすいソース・プログラムを書くということは，アセンブラだけに限らず，C，Pascal，FORTRANなどのすべての言語のプログラミングにおいて，最も重要なことなのです．

# 4 ソフトウェア開発手順とその実例

本章では，アセンブラによるソフトウェア開発を行う場合の主な4つの作業である，

・ソース・プログラムの作成(エディタ)
・アセンブラの実行(アセンブラ)
・ローダの実行(ローダまたはリンクローダ)
・デバッグ作業(デバッガ)

について，その一連の作業手順を解説しそれぞれの実例を示します．

これらの作業を行うには，それぞれ(　　)内に示した「道具」，つまりソフトウェアツールと呼ばれるプログラムを使わなければなりません．これらのツールは，同じ種類のものでも各メーカーによってその使い方が異なり，開発作業での具体的な操作法などにはある程度の違いがあります．

本章の実行例には，8ビットCPUのソフトウェア開発で，現在最も広く使われており，実務向けの環境も整っているCP/Mベースの各種のツール(5．1章参照)を使います．

ただし本章で問題にしているのは，それぞれの作業における各ツールの具体的な操作法やアセンブリ言語のプログラミング作法ではありません．ここで理解していただきたいのは，アセンブラによるソフトウェア開発の全体を通しての流れ，手順，および，それぞれのツールの働きなどであり，作成されるソース・プログラムや生成されるオブジェクト・プログラムなどについての基本的な知識なのです．

# 4/1 開発手順の基本

アセンブラによるソフトウェア開発を行うには，各種のソフトウェアツールが必要です．その主要なものを次に示します．

- **エディタ**――ソース・プログラムを書くためのツール
- **アセンブラ**――ソース・プログラムから，オブジェクト・プログラムを生成するためのツール
- **ローダ**――本来はその名のとおり，オブジェクト・プログラムをメモリ上にロード(格納)するためのツールであるが，現在の開発環境ではその意味合いが，だいぶ異なる．「ローダ」といえば，インテルHEX形式と呼ばれる特別な形式のオブジェクト・プログラムを，実行可能なオブジェクト・プログラムに変換したり，それとは別の形式の複数のオブジェクト・プログラムを結合して1本の実行可能なオブジェクト・プログラムなどを生成するためのツールを指すことのほうが多い．後者は普通「リンクローダ」とか「リンカー」などと呼ばれる．これらのオブジェクト・プログラムの形式については後述
- **デバッガ**――バグ取りツール．できあがったオブジェクト・プログラムが目的どおりに正しく動作するかをテストしたり，正常に動作しない場合はその原因を究明するための各種の機能を持ったツール

以上の4種類がアセンブラによるソフトウェア開発に必要な主要ツールです．これらのツールは，1つのソフトウェアを開発する場合でも，何度も繰

り返して使います．というのは，非常に簡単なプログラムを除けば，最初にできあがったプログラムはほとんどの場合どこかにミスがあり，正常に動作しないからです．そうなるとエディタによるソース・プログラムの修正からやり直さなくてはならないわけで，それを何回か，何十回か繰り返して行うのが普通です．

では，これらのツールを使ってどのような手順でソフトウェアの開発を行うのか，次の図で解説しましょう．図に示した例は，CP／Mに標準装備されているエディタやアセンブラなどの開発ツールだけを使った場合のものです．*

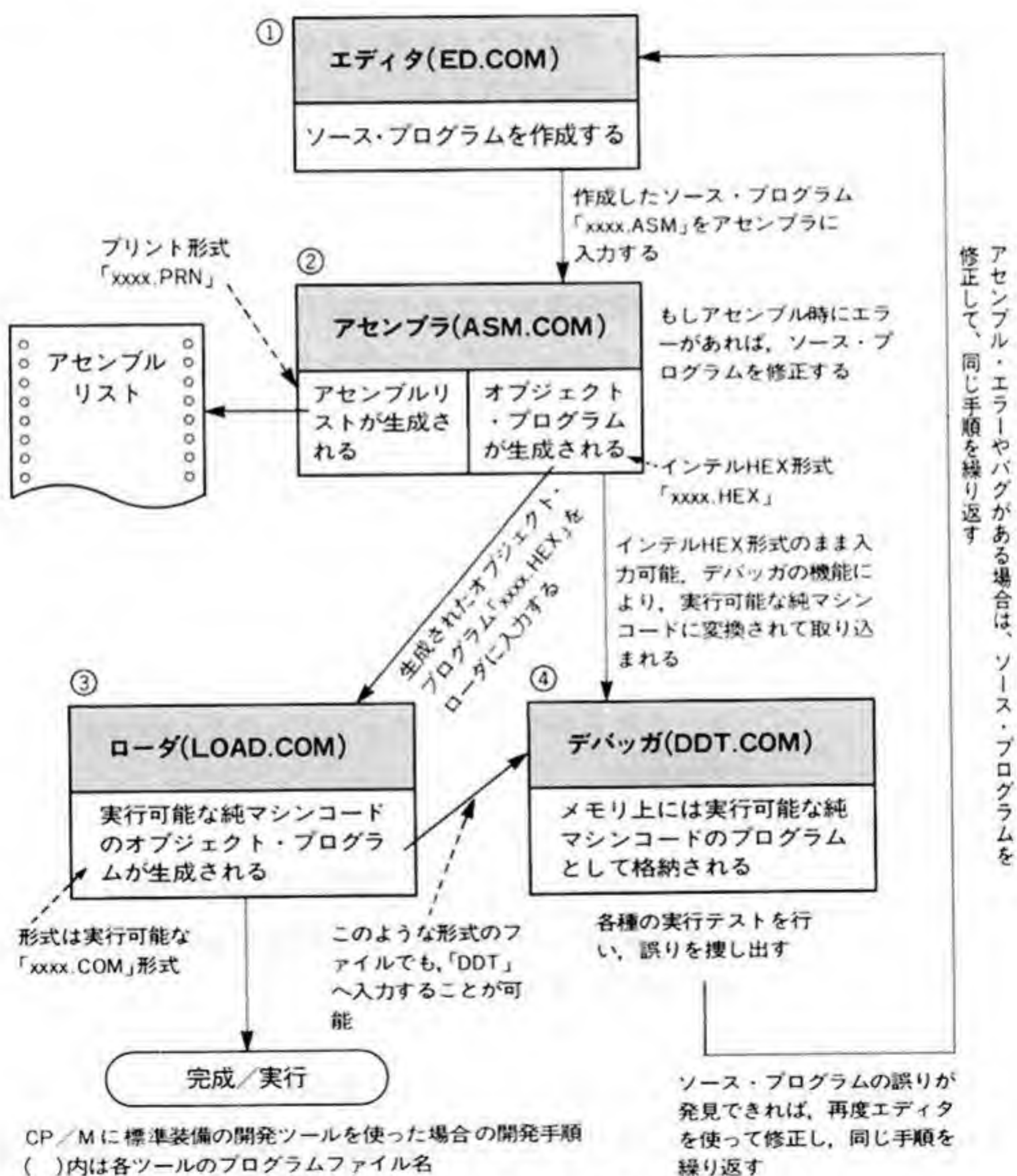



図4-1-1　アセンブラによるソフトウェア開発手順

図の手順①～④の流れの繰返しが，アセンブラによるソフトウェア開発の基本です．多くのパーソナル・コンピュータに標準装備されているBASICインタープリタによる開発手順とはだいぶ異なることに気がつかれたと思いますが，その相違点を次に示しましょう．

## BASICとの開発手順の相違

主な相違点として次の2つが挙げられます．

### (a) アセンブラはソース・プログラムのままでは実行できない

BASICはソース・プログラムをそのままの状態で実行できますが，アセンブラはソース・プログラムの状態のままで実行することはできません．アセンブリ言語のプログラムを実行するには，そのソース・プログラムを，CPUが実行可能なマシン語に変換しなければならないのです．この変換作業を行うためのツールがアセンブラです．このような形態の言語を，BASICのようなインタープリタ（解釈-実行形言語）に対して「コンパイラ」（翻訳-実行形言語）と呼びます．この両者の違いを，次の表にまとめておきます.**

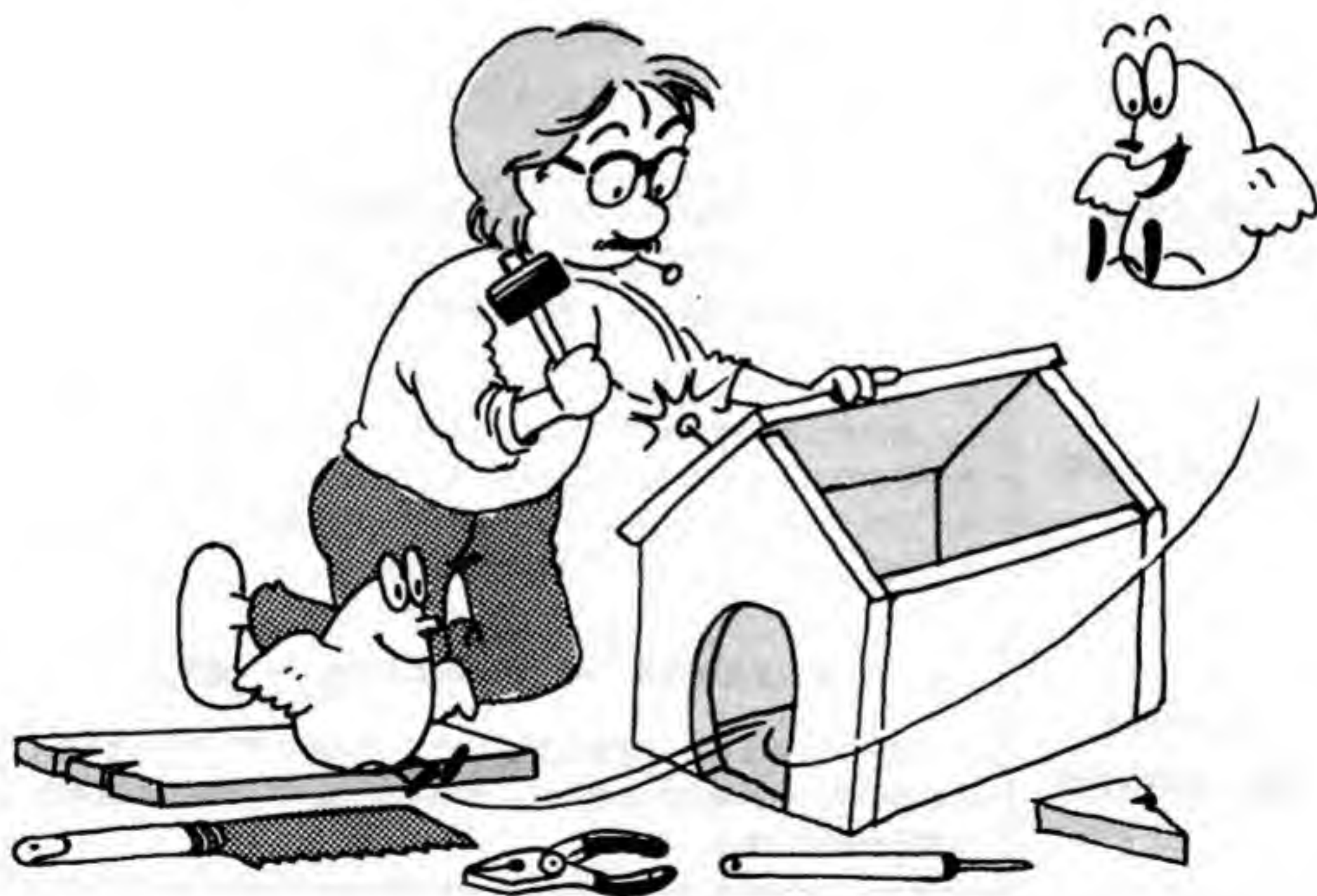
| 言語の形態 | ソース・プログラムから実行まで |
|---|---|
| インタープリタ<br>（解釈－実行形言語） | インタープリタ<br>ソース・プログラム → そのままインタープリタによって実行<br>ソース・プログラムを各ステートメント単位で，その場その場で解釈し，あらかじめ用意されているマシン語のルーチン（ある機能を持ったプログラム）を呼び出しながら実行していく．よって実行にはソース・プログラムとともに,BASICインタープリタ自身のプログラムが同時にメモリ上に存在していなければならない |
| コンパイラ<br>（翻訳－実行形言語） | コンパイラ<br>ソース・プログラム → マシン語に変換 → 直接実行<br>ソース・プログラムから完全に独立したマシン語のプログラムが生成されるので，何の助けも必要なく，そのマシン語のプログラムは単独で実行が可能である． |

図4-1-2 インタープリタとコンパイラの違い

*CP/Mの機能と重要性については5章で解説する．
**インタープリタやコンパイラに関しては，12章「アセンブラから高級言語へ」で詳しく解説する．

**（b） 各作業はそれぞれ専用のプログラムによって行う**

アセンブラでは，例えばソース・プログラムの作成にはエディタ，マシン語への変換（コンパイルと呼ぶ）にはアセンブラ，デバッグにはデバッガというように，それぞれ専用のソフトウェアツールで作業します．これに対してパソコンに組込みのBASICでは，BASICの中だけですべての作業が行えます．

開発手順に関するアセンブラとBASICインタープリタとの主な相違点はこの2点ですが，開発環境に関してはさらに大きな差があります．アセンブラのエディタやデバッグツールなどは，BASICに内蔵されているものに比べはるかに多機能で強力であり，実用的です．

# オブジェクト・プログラムの形式について

ここまでの説明では，「オブジェクト・プログラム」と「マシン語」をほとんど同じ意味で使ってきました．しかし正確にいうと，この両者は別のものです．マシン語はそのまますぐに実行できるものであり，オブジェクト・プログラムは，ソース・プログラムに対してアセンブラから生成されるものです．そしてソース・プログラムから生成されるオブジェクト・プログラムは，使用するアセンブラの種類によってその形式が異なります．代表的な3つの形式を，次の図に示します．詳しくはそのつど解説しますので，ここではただオブジェクト・プログラムにもいくつかの種類があることを認識してください．

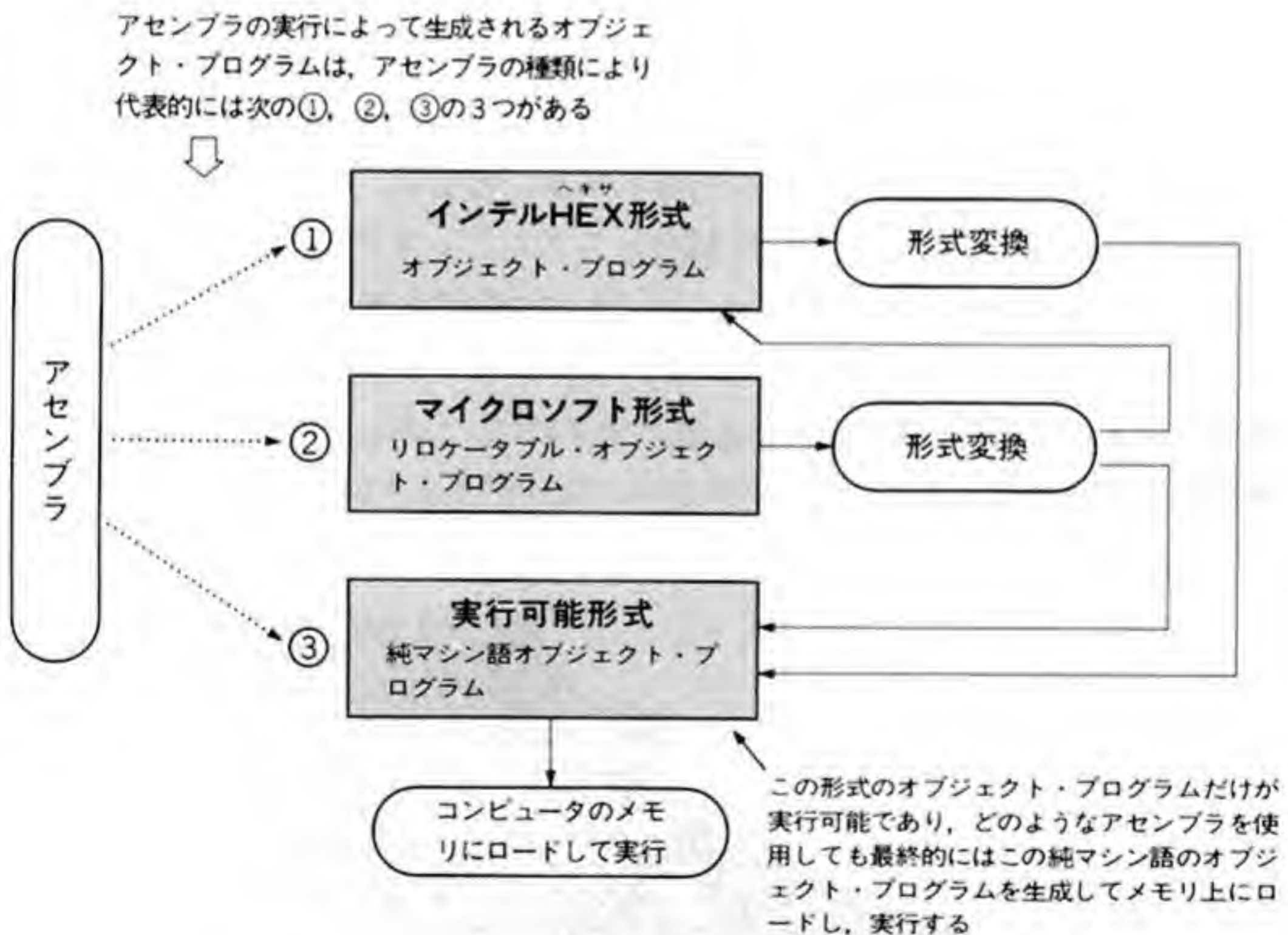



図4-1-3　代表的なオブジェクト・プログラムの形式

# 4/2 CP/Mによる開発実習

文章での説明だけでは具体的なイメージがわかないと思いますので，以降は実際にソフトウェアツールを使って簡単なプログラムを開発してみましょう．ここで例題とするプログラムは，2章，3章でおなじみのメッセージ出力のプログラムです．

開発ツールとしては，みなさんがZ-80や8085，8080などのソフトウェア開発を本格的に行う場合に必ず使うことになるであろうCP/Mと，それに標準装備されたツールのみを利用します．

5．1章で詳しく解説しますが，CP/Mのシステムディスクには，エディタ，アセンブラ，ローダ，デバッガ，その他の開発ツールが標準装備されていますので，他のソフトウェアツールを購入することなく，アセンブラによるソフトウェア開発が可能です．CP/Mのシステムディスクに，どのような種類のプログラムが含まれているのか，ディスクの内容の最も基本的なものを次にリストアウトして示します．ただし，各メーカーからそれぞれの機種用に発売されているCP/Mのシステムディスクには，この上にさらに各社独自の補助的なプログラム(ユーティリティー・プログラムと呼ぶ)がいくつか付属しています．

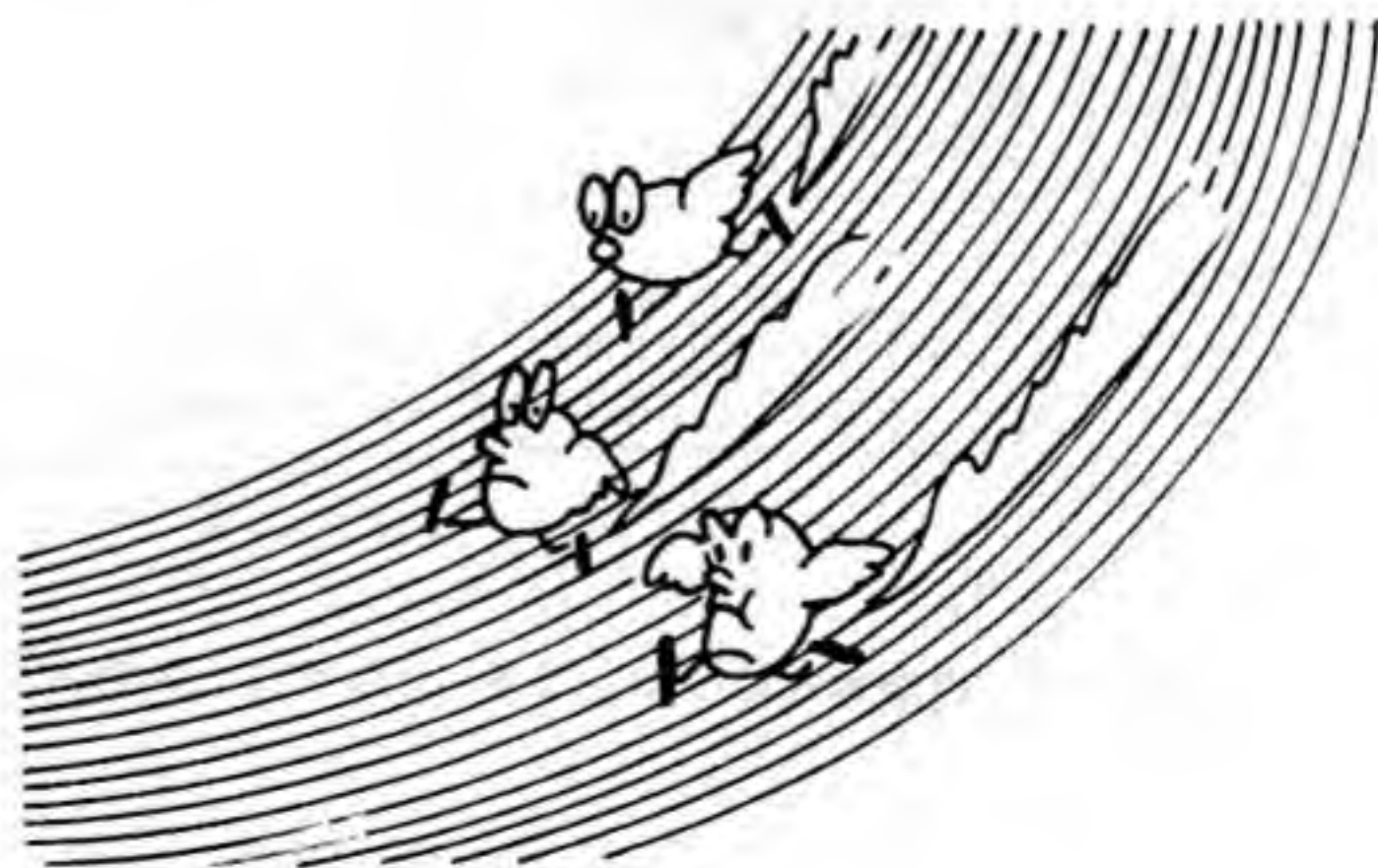



図4-2-1　CP/Mのシステムディスク上の標準的な各種のプログラム

```
A>STAT B:*.*
各プログラムファイルの大きさ(Kバイト)
                          プログラムファイル名
 Recs  Bytes  Ext Acc
   64     8k    1 R/W B:ASM.COM ……8080アセンブラ
   96    12k    1 R/W B:BIOS.ASM
   69     9k    1 R/W B:CBIOS.ASM
   38     5k    1 R/W B:DDT.COM ……8080デバッガ
   80    10k    1 R/W B:DEBLOCK.ASM
   49     7k    1 R/W B:DISKDEF.LIB
   33     5k    1 R/W B:DUMP.ASM ……メモリ・ダンプ・プログラムのアセンブリ・ソース・プログラム
    4     1k    1 R/W B:DUMP.COM ……メモリ・ダンプ・プログラム
   52     7k    1 R/W B:ED.COM ……エディタ
   14     2k    1 R/W B:LOAD.COM ……ローダ
   76    10k    1 R/W B:MOVCPM.COM
   58     8k    1 R/W B:PIP.COM ……周辺装置間のファイル転送/コピー・プログラム
   41     6k    1 R/W B:STAT.COM ……ファイルの状況報告などのプログラム(当リストの表示を行っ
   10     2k    1 R/W B:SUBMIT.COM     ているのは,このプログラム)
    8     1k    1 R/W B:SYSGEN.COM
   23     3k    1 R/W B:SYSGEN.HEX
    6     1k    1 R/W B:XSUB.COM
Bytes Remaining On B: 144k
                                      (ソフトウェア開発に関する主な
A>                                     ツールにのみ注釈をつけてある)
```

実習解説のための作業は,このリストにマークされているプログラム,「ED.COM」,「ASM.COM」,「LOAD.COM」,「DDT.COM」の4つの開発ツールで行います.ただしCP/Mのシステムディスク上のアセンブラ「ASM」とデバッガ「DDT」は8080CPU用のものですので,Z-80のニーモニック(命令語)を使ったソース・プログラムはアセンブルできません.そこで,Z-80のニーモニックを,そっくりそのまま8080CPUのニーモニックに置き換えたソース・プログラムを示しておきます.メッセージ出力の例題プログラムは,Z-80と8080に共通(コンパチブル)のCPU命令のみを使っていますので,各命令が1対1で置き換えられています.以前のZ-80のソース・プログラムとよく対比してみてください.CP/Mベースの別売のアセンブラやデバッガには,Z-80用はもとより,各種のCPUのものが用意されているのですが,ここではCP/Mのシステムディスクに含まれているツールだけを使います.これは,CP/Mが走る環境さえあれば別売ソフトを何も用意せずに実習できるよう考慮したからですが,そればかりでなく8080アセンブラは本格的なソフトウェア開発者にとっては常識といってよいほど基本的で必須の知識でもあるからです.ぜひ試みることをお勧めします.*

*8080アセンブラの重要性については,「はじめに」や「APPENDIX 3」にも述べてあるとおり.また,Z-80対8080,8085についてやそれに対応するツールについては,5.1章で詳しく説明する.

開発ツールが異なれば，当然その操作法も違ってきます．しかし，本章の冒頭でも述べたとおり，本章の目的は各ツールの表面的な操作法を学ぶことではありません．それよりも，これらのツールを使った各手順における処理の本質的なところに注目してください．

## エディタによるソース・プログラムの作成

私たちとコンピュータとの接点であり，すべてのソフトウェアを生み出す始点になるものがこのエディタです．実行できるプログラムを作るには，何といってもまず最初にソース・プログラムを書かなくてはなりません．

ここでの作業の目的は，エディタを使ってアセンブリ言語によるソース・プログラムを作成し，それをディスク(テープベースの場合はテープ)に「ソース・プログラム・ファイル」としてセーブすることです．アセンブリ言語のプログラミング作法や，その書き方などは，ここでは問題にしていません．

次の図は，エディタ・プログラムを実行してソース・プログラムを作成する場合の，ディスクとコンピュータとのデータのやり取りを示したものです．

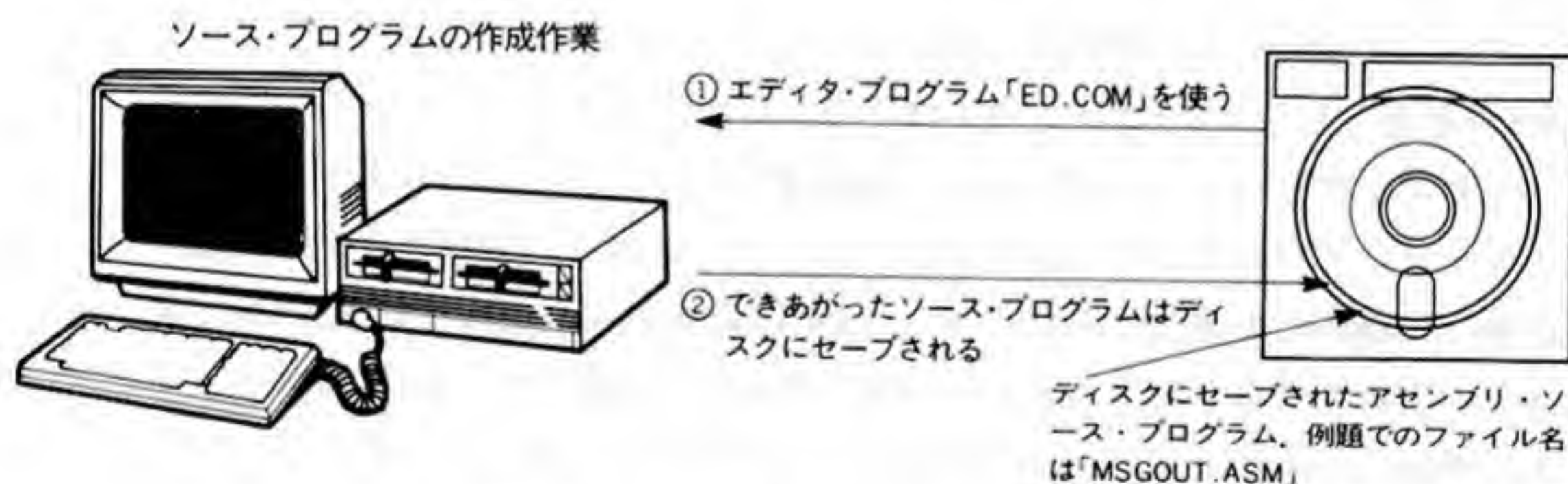



図4-2-2　ソース・プログラム作成作業の流れ

では，CP／Mのシステムディスクに付属のエディタ「ED」を使って，ソース・プログラムを作成する過程を具体的に実習解説していきましょう．

まずエディタ・プログラムを起動して，エディタの世界に入らなければなりません．このエディタのプログラム・ファイル名は，「ED.COM」です．

これを起動するには次のコマンドライン*をキー入力します.

ED　ファイル名↲

このコマンドを実行すると，左の図に示したように，ディスク上のエディタ・プログラムがコンピュータのメモリにロードされ，コマンドラインの「ファイル名」で指定されたファイルを作成するための準備が整います.

CP／Mに装備されているエディタ「ED」は，スクリーンエディタではなく，ポインタ形式のエディタと呼ばれるもので，この操作にはかなりの経験が必要で，自由に使いこなすようになるのにひと月やふた月はかかるでしょう（ポインタ形式のエディタに関しては，5．1章参照）.

しかし，このCP／Mのエディタは，ポインタ形式では最も使いやすく強力なエディタのひとつで，ポインタ形式のエディタの代表的存在でもあります.もし別売のスクリーンエディタを持っていても，いちおうは使えるようにしておくことをお勧めします．何よりも，コンピュータにおけるデータ編集の基本的な動作を知ることができるので，このエディタを使うことが，今後コンピュータと関係していく上で非常に重要な経験となるでしょう.

さて，エディタを起動した後，それぞれのエディタの操作法に従ってソース・プログラムを入力していきます．全部の入力が終わったら，もう一度内容をよくチェックし，入力ミスがあれば訂正して，エディタ・プログラムを終了します.

エディタを正常に終了すると，作成したソース・プログラムは自動的にディスクにセーブされますのでここでの作業はひとまず終りです.この時点で，ディスク上には，アセンブリ・ソース・プログラムがセーブされています.

*このような，コンピュータに対して何らかの実行指示を与える命令行を「コマンドライン」と呼ぶ.

この一連のソース・プログラム作成作業の実行例を次に示します．ソース・プログラムのファイル名は，「MSGOUT．ASM」としています．

図4-2-3　ソース・プログラム作成作業の実行例

```
A>ED MSGOUT.ASM

NEW FILE
   : *i
   1: ;-----------------------------;
   2: ;     MESSAGE OUT PROGRAM     ;
   3: ;   for 8080,8085 ASSEMBLER   ;
   4: ;-----------------------------;
   5: ;
   6: BDOS    EQU     0005H      ; system call entry point
   7: CR      EQU     0DH        ; Carriage Return code
   8: LF      EQU     0AH        ; Line Feed code
   9: EOS     EQU     00         ; End Of String code
  10: ;
  11:         ORG     100H       ; start             00H
  12: ;
  13:         LXI     H,MESG     ; get to            f message
  14: LOOP:                      ;                   cter pointer
  15:         MOV     A,M        ; get ch            e to output
  16:         ORA     A          ; end of            =00?)
  17:         RZ                 ; if 'Z'=1, end of this program
  18:         PUSH    H          ; save character pointer
  19:         CALL    CHRPUT     ; character out
  20:         POP     H          ; restore character pointer
  21:         INX     H          ; pointer goes up
  22:         JMP     LOOP       ; jump for next character
  23: ;
  24: ;------ 1 character out sub routine ------
  25: ;
  26: CHROUT:
  27:         MVI     C,2        ; CP/M system call
  28:         MOV     E,A        ;          console out
  29:         CALL    BDOS       ;                 function
  30:         RET
  31: ;
  32: ;------ string data area for message ------
  33: ;
  34: MESG:   DB      CR,LF,'Good Morning',CR,LF,EOS
  35: ;
  36:         END                ; list end
  37: ^Z
      *e

A>DIR MSGOUT.*

A: MSGOUT    ASM
A>
```

わざと1か所，誤った記述を入れてありますが，これでいちおうソース・プログラムが作成されました．次の段階はこのソース・プログラムをアセンブルする作業です．*

## アセンブラの実行

ここでの作業は，作成したソース・プログラムに対してアセンブラを実行することです．つまりソース・プログラムをアセンブルします．この結果，オブジェクト・プログラムやアセンブルリストが生成されます．

次の図は，アセンブル作業におけるディスクとコンピュータとのデータのやり取りを示したものです．

アセンブラを実行

①アセンブラ・プログラム「ASM.COM」を使う

アセンブル・リスト・ファイル

インテルHEX形式のオブジェクトファイル

②生成された2つのファイルはディスクにセーブされる

インテルHEX形式オブジェクトファイル．例題でのファイル名は「MSGOUT.HEX」

リストファイル．例題でのファイル名は「MSGOUT.PRN」

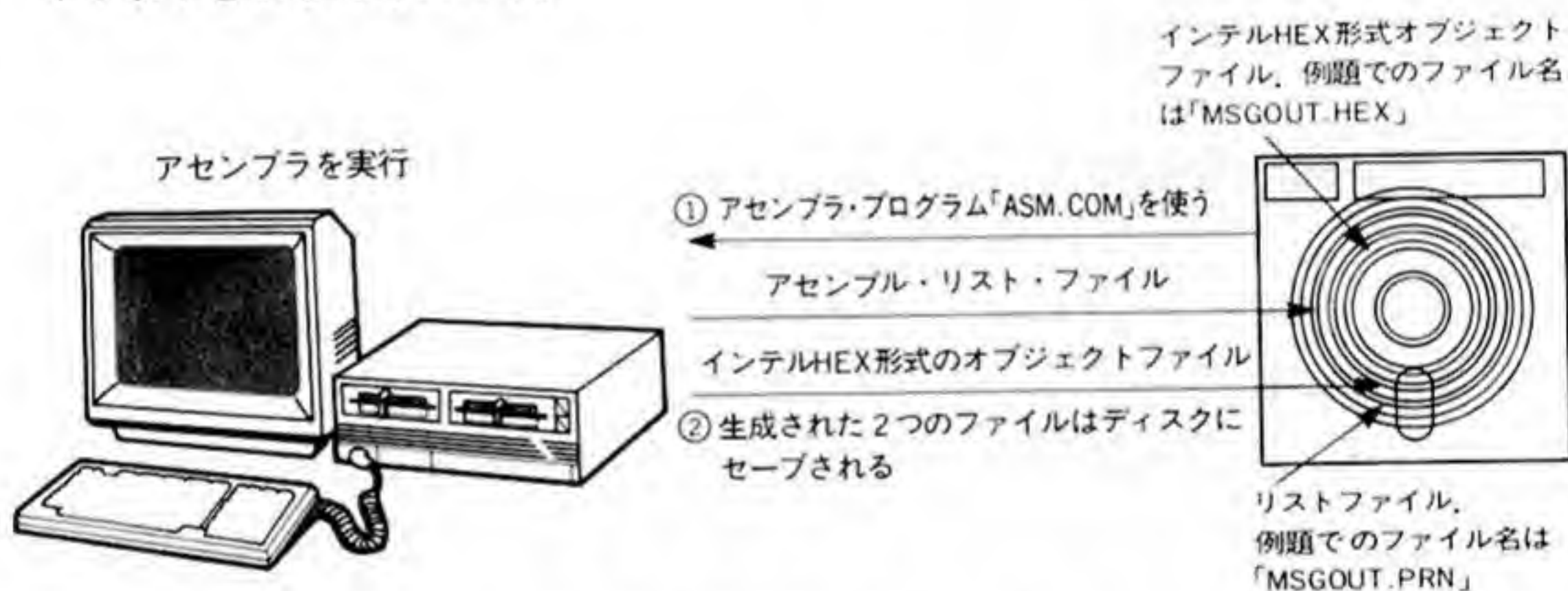



図4-2-4　アセンブル作業の流れ

アセンブルは自動的に行われるので作業は簡単です．前項のエディット作業と違って人間が考えなければならないような箇所はあまりありません．ソース・プログラムの中に文法上や記述上の誤りがなければ，ソース・プログラムに対してアセンブラを実行するだけでオブジェクト・プログラムへの変換は成功し，アセンブル作業は終了します．

ソース・プログラムに文法上，記述上の誤りがあればその部分のアセンブルは不可能です．その場合，アセンブラは，どの行にどのような誤りがあるのかをアセンブラの実行中や生成したアセンブルリスト上で指摘してくれます．

＊エディタ「ED」については5.1章を参照．

ただし，生成されたオブジェクト・プログラムを実行したとき，目的どおりの動作をするかどうかはアセンブルの成功とは別問題で，アセンブラの関知しない問題です．アセンブラにはただソース・プログラムを忠実にオブジェクト・プログラムに変換する機能しかありません．

ではCP/Mのシステムディスクに付属の8080用アセンブラ「ASM」を実行してみましょう．このアセンブラのプログラム・ファイル名は，「ASM .COM」であり，その実行は次のコマンドラインのキー入力によって行われます．

```
ASM ソースファイル名↵
```

ソースファイル名は「MSGOUT .ASM」ですが，実際のコマンドラインは，「. ASM」の部分を除いて，

```
ASM MSGOUT ↵
```

と入力します．このコマンド入力をすると，後はすべての処理が終わるまで自動的に運びます．

では，その実行例を示しましょう．ソース・プログラムに1か所誤りを入れてありますのでその部分がどのように処理されるかにも注目してください．

図4-2-5 アセンブラの実行例

```
A>ASM MSGOUT ↵  ……ソース・プログラム「MSGOUT.ASM」に対してアセンブラを実行する

CP/M ASSEMBLER - VER 2.0

U0107 CD0000          CALL    CHRPUT  ; character out  …アセンブル・エラーのある行が表示されている．先頭の文字「U」は，エラーの種類を表し，Undefindラベル・エラーの意味，つまり，このラベル「CHRPUT」が定義されていない

0127
000H USE FACTOR
END OF ASSEMBLY
  アセンブラの実行終了
A>DIR MSGOUT.* ↵  ……ファイル「MSGOUT」に関するディスク上のすべてのファイル名をタイプアウトする

A: MSGOUT   ASM : MSGOUT   PRN : MSGOUT   HEX
   ソース・プログラム  アセンブル・リスト・ファイル  インテルHEX形式のオブジェクト・プログラム
A>
                 （PRN, HEX：アセンブルより生成されたファイル）
```

アセンブル作業が終了し，オブジェクト・プログラムとアセンブルリストが生成されましたが，CRTディスプレイのメッセージは，ソース・プログラムにエラーがあったことを知らせています．生成されたアセンブルリストを見てみましょう．

図4-2-6　アセンブル・エラーが示されているアセンブルリスト

```
A>TYPE MSGOUT.PRN ↵ ……生成されたアセンブル・リスト・ファイルをタイプアウトする
                                  .
                                  .
                                  .
                   LOOP:                              ;        HL=character pointer
 0103 7E                        MOV      A,M          ; get character code to output
 0104 B7                        ORA      A            ; end of string? (A=00?)
 0105 C8                        RZ                    ; if 'Z'=1, end of this program
 0106 E5                        PUSH     H            ; save character pointer
U0107 CD0000                    CALL     CHRPUT       ; character out
 010A E1                        POP      H    ↑       ; restore character pointer
 010B 23                        INX      H○の誤り     ; pointer goes up
 010C C30301                    JMP      LOOP         ; jump for next character
                   ;
                   ;------ 1 character out subroutine ------
                   ;
                   CHROUT:
 010F 0E02                      MVI      C,2          ; CP/M system call
 0111 5F                        MOV      E,A          ;        console out

…アセンブルリストにも，同じようにアセンブル・エラーの種類別表示がされている．この行にUエラー(ラベル未定義エラー)
　がある．つまりラベル「CHROUT」が「CHRPUT」と誤って書かれている
A>
```

このようにアセンブラは，文法上，記述上のエラーを発見し，その行とエラーの種別をリスト上に表示します．

ではこの部分を，前項のエディタに戻って訂正し，再度アセンブルしてみましょう．その一連の実行例を次に示します．

図4-2-7 ソース・プログラムの訂正

```
( )内は，使用している編集機能のコマンドの種類を示しています

A>ED MSGOUT.ASM ↵ ……ソース・プログラム「MSGOUT.ASM」に対して，再度エディタを実行

   : *0A ↵ …………ソース・プログラムをメモリ上のエディット作業用のバッファに読み込む(a コマンド)
  1: *19:T ↵ …………ラインNo.19にキャラクタ・ポインタをセットして，その行をタイプアウトする(t コマンド)
 19:           CALL    CHRPUT   ; character out ……エラーのある行
 19: *SCHRPUT^ZCHROUT^Z0TT ↵ ……「CHRPUT」を「CHROUT」に書き換えて，その行をタイプアウトする(s コマンド)
 19:           CALL    CHROUT   ; character out ……「CHROUT」に書き換えられている
 19: *17::27T ↵ …………確認のためラインNo.17～27をタイプアウトする(t コマンド)
 17:           RZ               ; if 'Z'=1, end of this program
 18:           PUSH    H        ; save character pointer
 19:           CALL    CHROUT   ; character out
 20:           POP     H        ; restore character pointer
 21:           INX     H        ; pointer goes up
 22:           JMP     LOOP     ; jump for next character
 23:   ;
 24:   ;------ 1 character out subroutine ------
 25:   ;
 26:   CHROUT:
 27:           MVI     C,2      ; CP/M system call
 17: *E ↵ …………作業済のソース・プログラムをディスクにセーブして，エディタを終了する(e コマンド)

A>DIR MSGOUT.* ↵ ……ファイル「MSGOUT」に関するディスク上のすべてのファイル名をタイプアウトする

A: MSGOUT   ASM : MSGOUT   PRN : MSGOUT   HEX : MSGOUT   BAK
A>  新しいソース・プログラム          エディット作業を行う前のソース・プログラム
                                      が，バックアップファイルとして残されている
```

図4-2-8 再アセンブル

```
A>ASM MSGOUT ↵ …………再度アセンブラの実行

CP/M ASSEMBLER - VER 2.0 |
0127                     | アセンブル・エラーはない
000H USE FACTOR          |
END OF ASSEMBLY          |

  アセンブラの実行終了
A>DIR MSGOUT.* ↵ ……ファイル「MSGOUT」に関するすべてのファイル名をタイプアウトする

A: MSGOUT   ASM : MSGOUT   PRN : MSGOUT   HEX : MSGOUT   BAK
A>                  新しく生成されたものに書き換わっている
```

今回はエラーなくアセンブルが終了しました．ディスク上には，生成されたオブジェクト・プログラムと，アセンブルリストがセーブされています．つまりアセンブラを実行することでソース・プログラムから次の2つのファイルができあがったわけです．

| MSGOUT.ASM (ソース) | ⇨ | MSGOUT.HEX (オブジェクト) MSGOUT.PRN (リスト) |
|---|---|---|

次にこれらのファイルの内容を見ておきましょう．まずアセンブルリストをタイプアウトして示します．

図4-2-9 アセンブル・エラーなしのアセンブルリスト

```
A>TYPE MSGOUT.PRN

                   ;----------------------------;
                   ;     MESSAGE OUT PROGRAM    ;
                   ;   for 8080,8085 ASSEMBLER  ;
                   ;----------------------------;
                   ;
0005 =             BDOS    EQU     0005H   ; system call entry point
000D =             CR      EQU     0DH     ; Carriage Return code
000A =             LF      EQU     0AH     ; Line Feed code
0000 =             EOS     EQU     00      ; End Of String code
                   ;
0100                       ORG     100H    ; start address = 100H
                   ;
0100 211601                LXI     H,MESG  ; get top address of message
                   LOOP:                   ;         HL=character pointer
0103 7E                    MOV     A,M     ; get character code to output
0104 B7                    ORA     A       ; end of string? (A=00?)
0105 C8                    RZ              ; if 'Z'=1, end of this program
0106 E5                    PUSH    H       ; save character pointer
0107 CD0F01                CALL    CHROUT  ; character out
010A E1                    POP     H       ; restore character pointer
010B 23                    INX     H       ; pointer goes up
010C C30301                JMP     LOOP    ; jump for next character
                   ;
                   ;------ 1 character out subroutine ------
                   ;
                   CHROUT:
010F 0E02                  MVI     C,2     ; CP/M system call
0111 5F                    MOV     E,A     ;          console out
0112 CD0500                CALL    BDOS    ;                 function
0115 C9                    RET
                   ;
                   ;------ string data area for message ------
                   ;
0116 0D0A476F6FMESG:       DB      CR,LF,'Good Morning',CR,LF,EOS
                   ;
0127                       END             ; list end
A>
```

DB擬似命令によるデータのオブジェクトは，5バイト分表示され，残りは省略されている

ロード・アドレス

オブジェクト・プログラム

アセンブリ・ソース・プログラム

ここで使用したアセンブラ「ASM」は，CP/Mのシステムディスクに含まれている8080CPUのアセンブラですが，生成されたオブジェクト・プログラム(リスト左側)は，2章，3章でおなじみのZ-80でのそれとまったく同じであることがわかります．

これは，もとのZ-80のソース・プログラムでZ-80専用の命令を使わず，オブジェクトコード(マシンコード)やその働きが8080と共通の命令だけを使っているためです．Z-80用のニーモニックをそっくり8080用のニーモニックに置き換えたわけですから，当然のことともいえます．

次に，生成されたオブジェクト・プログラムのファイルを見てみましょう．アセンブラ「ASM」は，前述のインテルHEX形式のオブジェクト・プログラムを生成します．この形式は，オブジェクトコードをアスキーコードで表現する方式ですから，生成されたファイルは一種の「文章ファイル」*です．よって，通常の文章ファイルをタイプアウトして読むように，この形式のオブジェクト・プログラム・ファイルは「読む」ことが可能です．

生成されたオブジェクト・プログラムのファイルは，「MSGOUT．HEX」というファイル名でディスクにセーブされていますので，これをタイプアウトしてみましょう．

図4-2-10 生成されたインテルHEX形式のオブジェクト・プログラム

```
A>TYPE MSGOUT.HEX ↵
:10010000211601 7EB7C8E5CD0F01E123C303010E1F
:10011000025FCD0500C90D0A476F6F64204D6F72F5
:070120006E696E670D0A0015
:0000000000
A>
```



## オブジェクト・プログラムのロードと実行

さて次の作業として，生成されたインテルHEX形式のオブジェクト・プログラムを，CPUが実行可能な純マシンコードのオブジェクト・プログラムに

*本書では，一般に「アスキーファイル」とか「テキストファイル」といわれるものを，「読める」，「目に見える」という意味で「文章ファイル」と記述している．

変換して，できあがったプログラムをまず実行してみましょう．

ここでの作業に使用するツールである「ローダ」は，必ずアセンブラと対になって，各ソフトウェア・メーカーから提供されるものです．例えばCP/Mのシステムディスクに付属しているものであれば，アセンブラ「ASM」とローダ「LOAD」とが，対であり，マイクロソフト社のマクロアセンブラであれば，アセンブラ「M80」とローダ「L80」とが，対になっています．

ここで使用するCP/Mのローダ「LOAD」のプログラムファイル名は「LOAD.COM」であり，これはインテルHEX形式のオブジェクト・プログラムを，CPUが実行できる純マシンコードに変換する機能を持っています．

次の図は，ロード作業におけるディスクとコンピュータとのデータのやり取りを示したものです．

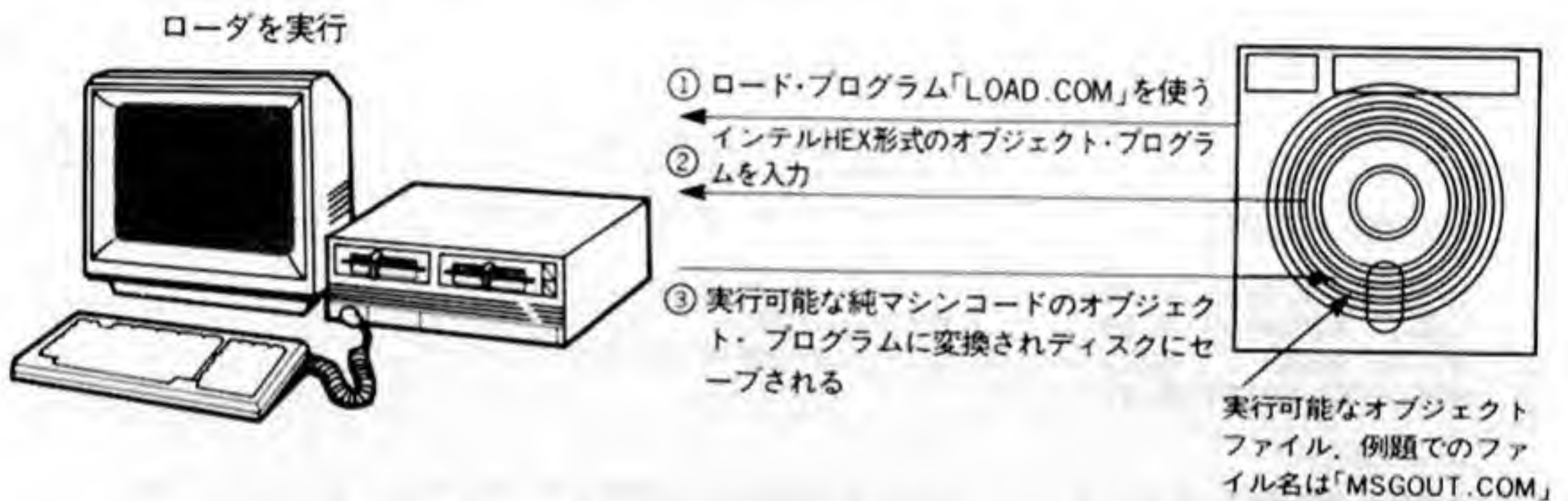



図4-2-11 インテルHEX形式→純マシンコードへの変換作業の流れ

この図は，ロード・プログラムを実行することにより，インテルHEX形式のオブジェクト・プログラムが，実行可能な形式である純マシンコードのオブジェクト・プログラムに変換されて，ディスクにセーブされる過程を示しています．

インテルHEX形式のオブジェクト・プログラムは，さきほど述べたように実行可能なオブジェクト・プログラムを「文章ファイル」の形式で表したものですので，CPUが実行できる純マシンコードそのものではありません．

読者の中には，なぜアセンブラから直接に，実行可能な純マシンコードのオブジェクト・プログラムを生成しないのか，またなぜインテルHEX形式のオブジェクト・プログラムなどという面倒なものが存在するのか，などの疑

問を持った方も多いと思います．確かに，この例のようにただこれだけの作業ですべてが終わるのなら，アセンブラから出力されるオブジェクト・プログラムの形式を，最初から純マシン語のプログラムにしておけばよいし，そのほうがアセンブラにとっても簡単です．しかし，いろいろな理由から，* わざわざ特別な形式のオブジェクト・プログラムに変換して出力しているのです．

さて，CP/Mのロード・プログラムの実行は簡単です．ロード・プログラムのファイル名は「LOAD.COM」で，実行するには次のコマンドをキー入力するだけです．

```
LOAD　オブジェクトファイル名↵
```

ではオブジェクトファイル「MSGOUT.HEX」に対するLOADプログラムの実行例を示します．

図4-2-12 インテルHEX形式→純マシンコードへの変換実行例

```
A>LOAD MSGOUT ↵ ……………「MSGOUT.HEX」に対してローダを実行

FIRST ADDRESS 0100 ……………スタート・アドレス0100H
LAST  ADDRESS 0126 ……………最終アドレス 0126H
BYTES READ    0027 ……………プログラムのバイト数 27H バイト
RECORDS WRITTEN 01

ローダの実行終了
A>DIR MSGOUT.* ↵ ……………ファイル「MSGOUT」に関するすべてのファイル名をタイプアウトする

A: MSGOUT   ASM : MSGOUT   PRN : MSGOUT   HEX : MSGOUT   BAK
A: MSGOUT   COM
    生成された実行可能な純マシン
A>  語のオブジェクト・プログラム
```

この操作で新たに生成されたファイル，「MSGOUT.COM」が実行可能な純マシンコードのプログラムファイルであり，これはメモリにロードしてそのまま実行することができます．

CP/Mは，ファイル名に「.COM」がついたディスク上の実行可能なプログラムであれば，特別なツールを必要とせずに実行できます．

実行するときは，実行しようとするプログラムファイル名の［.］より左

*特別な形式のオブジェクト・プログラムが必要な理由については，5.1章「アセンブラおよびローダ」の項で簡単に解説している．

の部分，例えば「MSGOUT.COM」なら，「MSGOUT」だけをキー入力し，リターンします．ただこれだけの操作で，MSGOUTのマシン語プログラムがディスクからコンピュータのメモリ上にロードされ，自動的に実行されます．つまり実行可能なオブジェクト・プログラムをメモリ上に移す「ロード」は，先ほどの「LOAD」プログラムには関係なく，CP/Mが自動的に行うわけです．

このような小さなプログラムは例外として，できあがった実行可能なプログラムには，必ずといってよいほどバグがあります．ソフトウェア開発の本来の順序からいうと，ここでそのデバッグ作業を行うのですが，今回はできあがったこのマシン語のプログラムをとにかく実行してみましょう．

では実行例を次に示します．

**図4-2-13 マシン語プログラムの実行**

```
A>MSGOUT ↵ ……………プログラム「MSGOUT.COM」の実行
Good Morning | 8080アセンブラにより作成されたオブジェクト・プログラムの実行によるメッセージの表示
A>
```

エラーがないことは最初からわかっていますので感激はしないでしょうが，めでたく目的どおりの動作をすることが確認されました．

## デバッグ作業

エラーなくアセンブルが成功しオブジェクト・プログラムができあがったとしても，それはソース・プログラムの文法上および記述上の誤りがないだけであり，これを実行したとき，目的どおりの動作をするかどうかはわかりません．前項でも述べたように，最初はほとんどの場合完動しないものです．そこでデバッグ(de-bug)作業を行うことになるわけですが，マシン語のプログラムをデバッグするのは，優れたデバッグツールを使ったとしてもかなりたいへんな作業になるでしょう．

プログラムが目的どおりに動かないとき，それがいわゆる高級言語で書かれたプログラムであれば，そのソース・プログラムの「文章」を見直すこと

になりますが，アセンブラの場合は，「文章」はもちろんCPUの各レジスタの動きまでを一つひとつチェックしなければなりません．

よってこれらの作業は，アセンブラの知識はもとより，マシン語によるCPUの働きを十分に理解していなくてはできないことになります．つまりデバッグ作業は，マシン語やアセンブラを始め，コンピュータに関する知識を総動員する必要があるのです．

さて，デバッグに関しての実習解説ですが，これにはアセンブラに関する総合的な知識が必要ですし，かなり多くの事柄がありますので別に章を設け，10章で解説します．

以上，アセンブラによるソフトウェア開発の基本的な手順や過程を一通り見てきました．本章までの内容で，アセンブラというものの全体の姿がいちおう理解できたのではないでしょうか．

# 5 ソフトウェア開発ツールとその機能

ソフトウェアを作成するには，いろいろな作業をするための道具となるプログラム（ソフトウェア開発ツール）が必要であることは，前章で実習解説したとおりです．

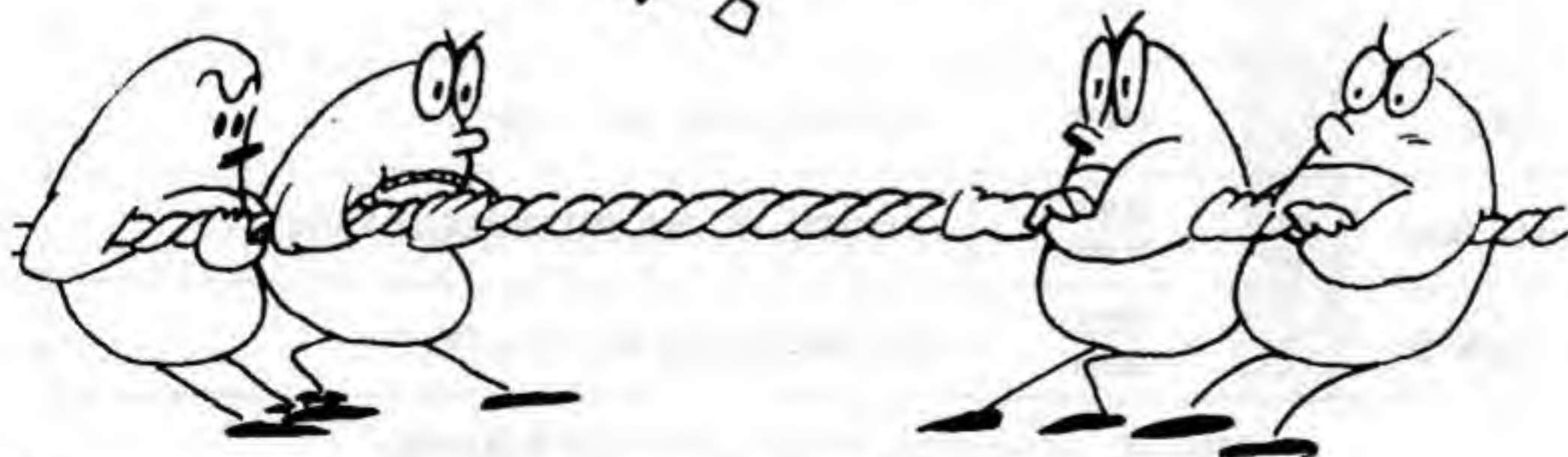
前章ではこの道具として，CP／Mのシステムディスクに含まれている開発ツールを使用して開発実習を行いましたが，実際多くの開発現場では別売のさらに強力な各種のツール（CP／M上で利用できるもの）が使われています．いずれにしても，Z-80や8085，8080などの8ビットCPUの本格的なソフトウェア開発には，CP／Mをベースにしたツールを使うことが一般的です．そこで本章では，まずCP／Mとは何かについて概説し，続いてCP／Mをベースにした各種の代表的な開発ツールを紹介しましょう．

一方，学習用とかあまり規模が大きくないプログラムの開発用には，CP／Mをベースとしない簡易版のツールも各社から発売されています．ここではディスク・ベースの「DUAD」と，学習用として作られたカセット・ベースの「MF ASM」の2つを紹介します．

# 5/1 CP/Mをベースにした各種のツール

CP/Mのシステムディスクに標準装備されたアセンブラやデバッガは，8080CPU用のもので，機能もそれほど高くはありません．そこで多くの人は，Z-80CPUに対応できてしかも多くの機能を持つ，各社から別売されているCP/M上の各種ツールを使っています．

Z-80や8085，8080などの8ビットCPUのソフトウェア開発ツールは，何といってもCP/Mベースのものが主流で，機能が高く使いやすいものが豊富にそろっています．つまり8ビットCPUを対象にする場合には，CP/Mをベースにすることが，完備された開発環境を最も安価に実現させる手段なのです．*

CP/Mや，16ビットのパーソナル・コンピュータのほとんどに採用されているMS-DOSなどは，「OS」（オペレーティング・システム）と呼ばれています．この「OS」は，コンピュータ・システムの中で非常に重要な位置を占めている「基本ソフトウェア」です．ソフトウェア技術を修得するには，アセンブラを始めとする各種の言語によるプログラミングを学ぶことと同時に，この「OS」の概念とその使い方を知ることが必要不可欠です．

＊同じ8ビットCPUでも6809 CPUについてはOS-9というOSが主流．

本章ではOSについて多くのページを割くことができませんが，その概要を解説しておきます．次項のタイトルは，「CP／Mとは」ですが，ここではCP／MそのものではなくOSの概念を理解することが大切です．

## CP／Mとは

CP／Mのマニュアルとシステムディスク

CP／MはOSです．MS-DOSやUNIXもOSです．普及は今一歩ですが，FLEXとかOS-9といったOSもあります．たぶんみなさんも，これらのOSの名前は，見たり聞いたりしたことがあるでしょう．

次の表に，これらのOSの特徴や，それがどのようなコンピュータに使われているかなど，その概要を示しておきます．

| OSの種類 | 適合CPU | | そのOSが用意されている代表的機種 |
|---|---|---|---|
| CP/M | 8ビット | 8080, 8085<br>Z-80 | PC-8001, PC-8801, MZ-2000, X1, PASOPIA, FM-7, SMC-70, SMC-777, IF800, QC-10 |
| OS-9 | 8ビット | 6809 | レベル3, FM-8, FM-7, FM-11 |
| FLEX | 8ビット | 6809 | レベル3, FM-8, FM-7, FM-11 |
| MS-DOS | 16ビット | 8088<br>8086 | PC-9801, PC-100, PASOPIA16, MULTI16, JX |
| CP/M-86 | 16ビット | 8088<br>8086 | PC-9801, N5200, MULTI16, FM-11 |

図5-1-1　パーソナル・コンピュータ上の各種のOS

これらのOSはすべて，ディスクを装備したコンピュータ・システムのためのものなので，「DOS」(ディスク・オペレーティング・システム)とも呼ばれます．CP/Mは，これらのOSの中で，80系(8080，8085，Z-80，μPD780など)のCPUを使った8ビット・マイクロコンピュータの，実質的な標準OSとなっています．

現在80系のパーソナル・コンピュータは，ほとんどのものがCPUにZ-80を使用しBASICマシンとして売られていますが，これらの製品は，学習用マシンなどの低価格のものは別として，国産・輸入を問わずほとんどすべてにそれぞれの機種用のCP/Mが用意されています．ユーザーは，それぞれの機種のメーカーから発売されている(あるいは別のソフトウェア・メーカーから発売されている)各機種用のCP/Mのシステムディスクを購入すればよいわけです．

さて，CP/Mは8ビット・マイクロコンピュータ用の最も普及しているOSであることはわかりましたが，それではOSとは何でしょう．OSは，それだけでは私たちユーザーにあまり多くのことをしてくれません．せいぜいディスク上のファイル名の表示であるとかファイルのコピーや削除などの機能が目につく程度です．OSはソフトウェアですが，私たちが最終的に利用している，各種のビジネスソフトや，プログラミングのための言語や，ゲームソフトなど(これらのソフトウェアを総称して，ユーザーソフトと呼ぶことにしましょう)とは用途が異なります．

これらのユーザーソフトに対してOSは，「基本ソフトウェア」であり，コンピュータのハードウェアに密着してコンピュータの最も基本的な動作を可能にします．

キーボードからの文字の入力，CRTディスプレイ上への表示，ディスク上のファイル管理，ディスクへのデータの書込みと読出し――こういったコンピュータの基本的な動作を私たちユーザーに提供し，かつコンピュータ・システム全体の働きを管理するソフトウェアが，このOSであるわけです．

### ＊コンピュータ・システムの中でのOSの位置

次に，コンピュータ・システムの中でOSが占める位置を図で示しましょう．私たちがコンピュータを使って何かの仕事をしているとき，コンピュータと

ユーザーの間では,

- **コンピュータのハードウェア**
- **OS**
- **ユーザーソフト**
- **ユーザー**

が,次の図のように関係して,システム全体が動作しています.

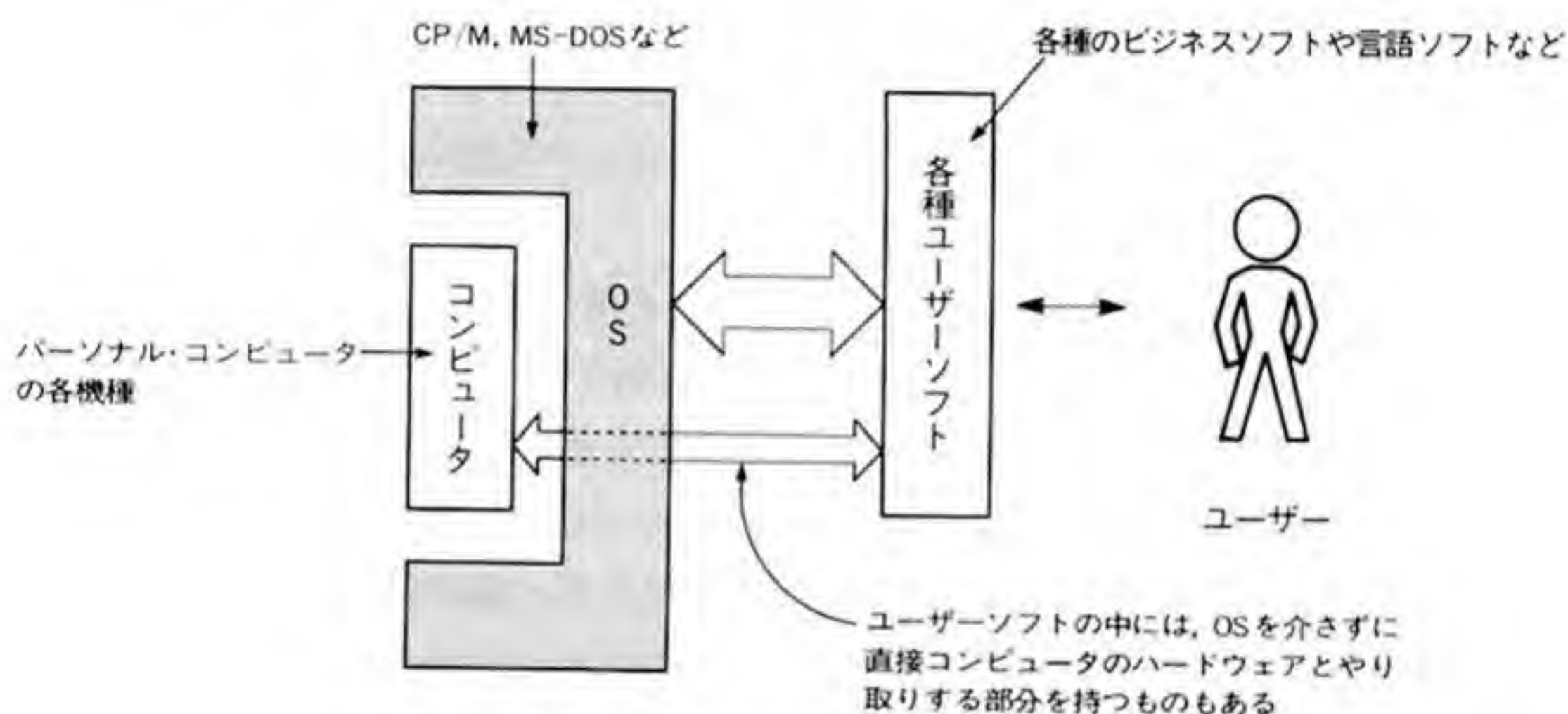



**図5-1-2　コンピュータ,OS,ユーザーソフト,ユーザーの関係**

この図に示したOSの位置に注目してください.

よくいわれることですが,コンピュータ・システムは「ソフトウェアがなければただの箱」です.ではコンピュータはユーザーソフトさえあれば働くのでしょうか.これだけでは働きません.図でわかるように,ユーザーソフトは,OSの上に載せてこそ動作するのです.*

例えば市販のワープロソフトの場合を考えてみましょう.ワープロソフトには,それを動作させる環境(走行環境)に2種類の形態があります.ひとつはMS-DOSやCP/Mなどの流通OS(広く世間に普及しているOSのことをいう)上で動作する形態,もうひとつはそれらの流通OSを利用せず,マシン

*ただし1章でも述べたように,洗濯機などに組み込まれる機器制御用のマイコンは,OSに依存せずに動作する.

に組込みのBASIC上で動作する形態です(とはいっても，そのプログラムがBASIC言語で書かれているわけではない)．後者の形態は，「スタンドアローン」* と呼ばれています．

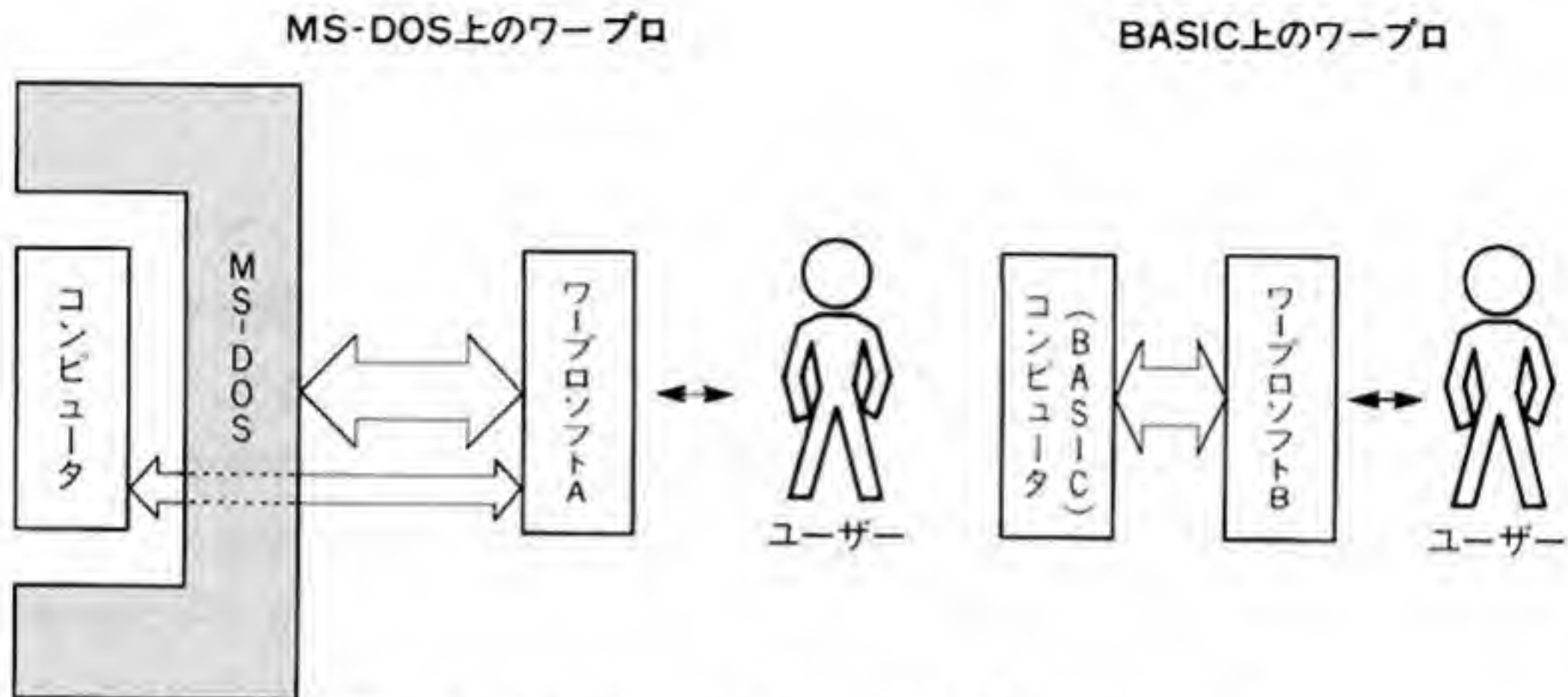



図5-1-3　MS-DOS上のワープロとBASIC上のワープロ

スタンドアローンであればOSなどいらないではないかと思われた方がいるかも知れません．しかし注意しなければならないのは，「BASICにはOSに相当する部分が含まれている」ということです．このことは，ほとんどのBASICの参考書が触れていないことですが，本書の読者クラスには理解しておいてほしいBASICに関する重要な知識です．つまり，私たちが今まで何も考えることなしに「パソコン=BASIC」として扱ってきたBASICは，

- **OS部**
- **エディタ部**
- **BASICインタープリタ本体部**

から成り立っているのです．

つまり，今までOSなどまったく意識することのなかったBASICのプログラムも，実はBASICに組み込まれているOS部によって，コンピュータのハードウェアと結ばれ動作していたのです．

* 自分だけで動作するという意味．

流通OSを利用していない，スタンドアローンのワープロソフトなどがBASICを利用しているといっても，そのプログラムがBASIC言語で作られているわけではありません．ワードプロセッサのような高度でかつ大規模なソフトウェアになると，BASIC言語を使っていたのでは実用的なものを作ることは不可能です．ほとんどのものはアセンブラやCといったコンパイラ形のプログラミング言語（12章参照）によって書かれています．

CP／MやMS-DOSのような流通OSを利用しない，スタンドアローンのソフトウェアにも，やはりOSは必要です．スタンドアローンで動作するソフトウェアは，これをマシン組込みのBASICのOS部で代用しています．「BASICを利用している」というのは，「そのOS部を利用している」にすぎないわけです．ただしBASIC内のOS部の機能は，MS-DOSなどに比べるとかなり貧弱なので、不足している機能は独自に用意しています．例えばワープロソフトであればそのプログラムの中に，不足しているOS部の機能を追加しなければなりません．この様子を次の図に示しましょう．前図と比較してください．

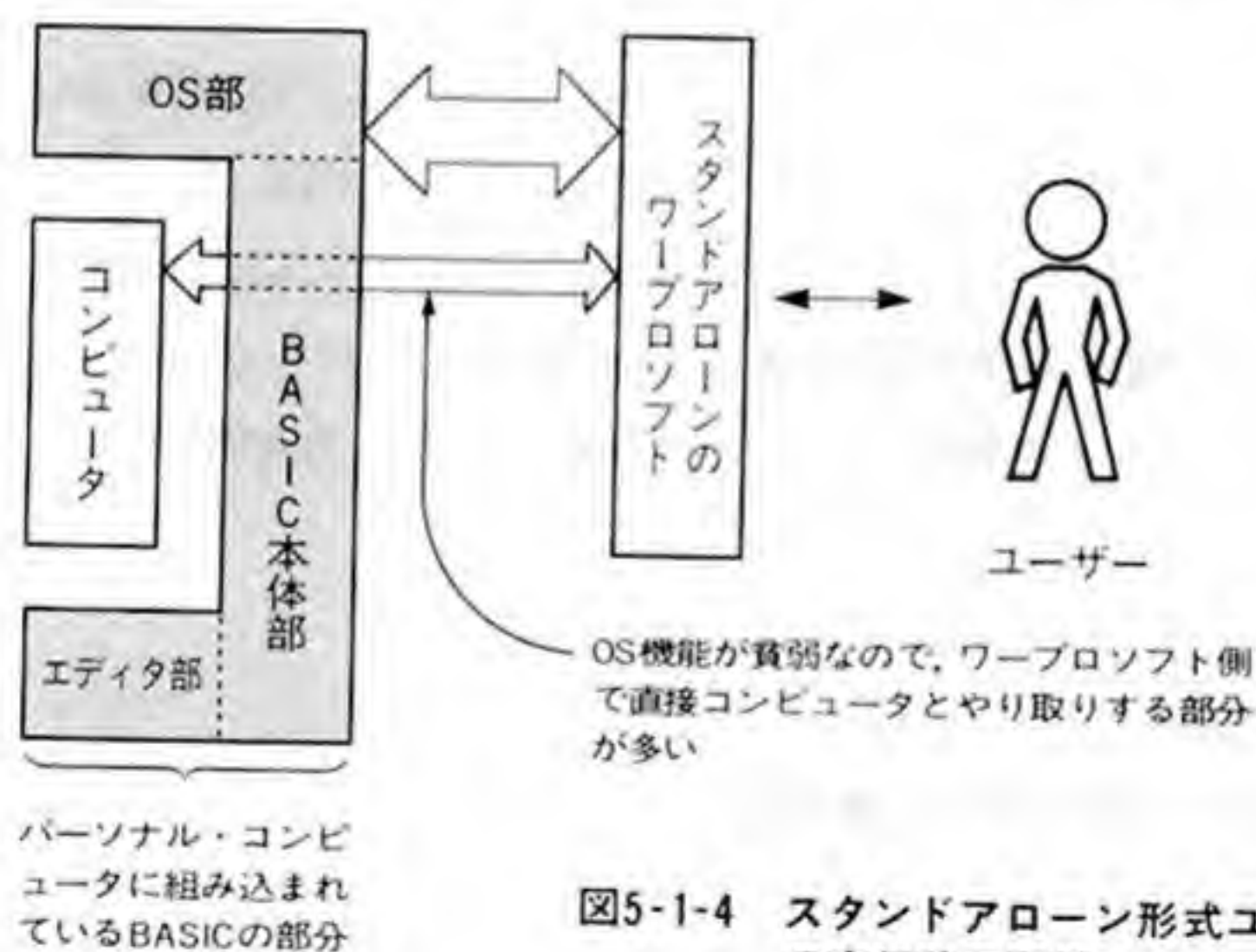



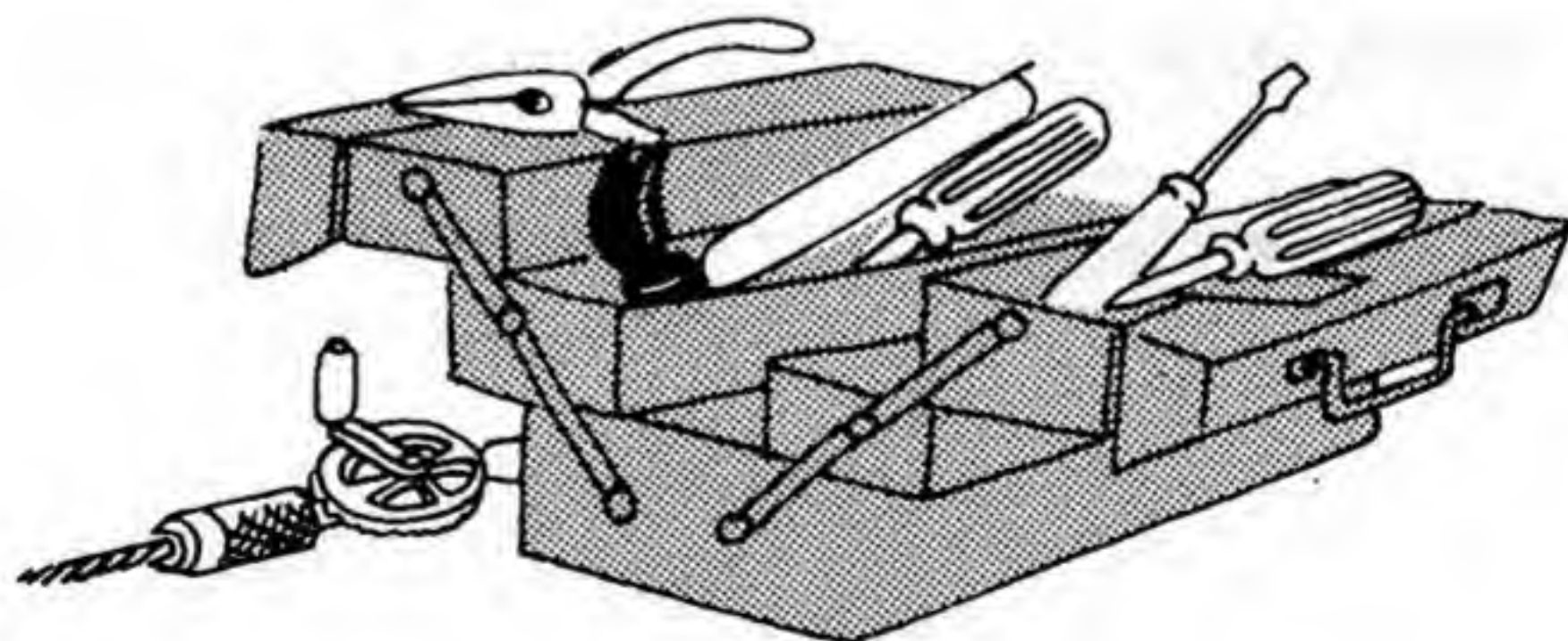
図5-1-4　スタンドアローン形式ユーザーソフトの実行時の関係

このように，流通OSを利用しない場合でも，ユーザーソフトとコンピュータのハードウェアの間には，かなり貧弱ですが，やはりOSに相当する「基本ソフトウェア」の部分が存在しています．

OSの概念は，今後コンピュータを本格的に応用していく上で非常に重要なものとなります．実務用のコンピュータ・システムは，「コンピュータ+OS」が大前提であったのです．このようなOSの存在と重要性をよく認識しなければ，これからのコンピュータの世界は広がりません．

パーソナル・コンピュータのユーザーが，今後とも使用することになるであろう主流OSは，8ビット・マシンではCP/Mであり，16ビット・マシンではMS-DOSです．そしてさらに高度なOSとしてはUNIXなどが用意されています．

OSがこれほど重要なものであるにもかかわらず，今までのBASICのユーザーに，それもDISK BASICのユーザーにさえ，まったくといってよいほど認識されていなかったことは，たいへん不幸なことです．

残念ながら本書ではこれ以上OSに関してページを割くことができませんが，いずれにしても本格的なソフトウェア開発を目指す方は，OSに関して別の参考書でしっかり学ぶ必要があります．これらの知識や実際の操作法を知らずに，本格的なソフトウェア開発は不可能です．拙著も含めてOSに関する代表的な参考文献を欄外に挙げておきます．*

＊CP／M　：『入門CP／M』，『実習CP／M』，『応用CP／M』
MS-DOS：『入門MS-DOS』，『実用MS-DOS』，『標準MS-DOSハンドブック』
（いずれもアスキー出版局発行）

### ＊本書の実行例でよく使われるCP/Mのコマンド

本書ではソフトウェア開発の実行例の多くを，CP/M 上で行っています．CP/M 自身に備わっているコマンドのうち，次に示す2つを多用しますので，それらを簡単に紹介しておきましょう．

```
DIR  ――(ディレクトリ・コマンド)
TYPE ――(タイプ・コマンド)
```

DIR コマンドは，ディスク上にどのようなファイルがセーブされているかを調べるコマンドです．DISK BASIC では，FILES コマンドに相当します．CP/M のプロンプト(コマンドの入力を促すために入力を受け付ける記号)は，「A>」とか「B>」ですので，A>の後に，

```
A>DIR ↵
```

と入力すれば，ドライブA(ドライブ1)上のディスクにセーブされているすべてのファイルのファイル名を表示することができます．さらに[＊](アスタリスク)記号を使うと，

```
A>DIR A＊.＊↵   ― ファイル名の頭が「A」であるすべてのファイル名を表示する
A>DIR AB＊.＊↵  ― ファイル名の頭の2文字が「AB」であるすべてのファイル名を表示する
A>DIR ＊.COM↵   ― ファイル・タイプ*が「COM」であるすべてのファイル名を表示する
```

＊CP/Mのファイル名の形式は，8文字までの主ファイル名と，3文字までの副ファイル名（ファイルタイプとか，エクステンションとか呼ぶ）から成っている．例えば「ABCDEFGH.XYZ」などで，副ファイル名の前には，ピリオド[.]を置く．

このように，ある条件に合致したファイル名だけを表示することができます．このような［＊］記号の使い方を，「ファイルマッチ」とか「ワイルドカード」とか呼んでいます．

TYPEコマンドは，指定された任意の文章ファイルを表示するためのコマンドです．BASICにはこれに相当するコマンドはありません．強いていえば，LOADコマンド＋LISTコマンドですが，TYPEコマンドは，単に文章ファイルを表示するだけです．例えば，「ABCD．ASM」という文章ファイルをタイプアウトするには，

```
A>TYPE  ABCD．ASM ↵
```

と入力します．

## 代表的なツール

ではまず，8ビットのCP／MをベースにしたZ-80，8085，8080CPUのアセンブラによるソフトウェア開発ツールのうち，代表的なものを一覧表にして示しましょう．いずれも一般的に使われているもので，ソフトウェア販売店から容易に入手できます．

| ツール | 製品名 | メーカー | 備　　考 |
|---|---|---|---|
| エディタ | ED (イーディー) | デジタルリサーチ | ポインタ形式のエディタ<br>CP／M システム・ディスクに標準装備 |
| | Word Master (ワード・マスター) | マイクロプロ | スクリーンエディタ.CP／MのEDとコマンド・コンパチブルのポインタ形式モードも含む |
| アセンブラ(ローダ) | ASM (エーエスエム)<br>(LOAD) | デジタルリサーチ | 8080アブソリュート・アセンブラ. インテルHEX形式のオブジェクトを出力. CP／M のシステムディスクに標準装備 |
| | MAC (マック)<br>(LOAD) | | 8080, 8085, Z-80共用マクロアセンブラ. Z-80ニーモニックは, ザイログ表記ではなくインテル拡張表記である. インテルHEX形式のオブジェクトを出力する |
| | RMAC (アールマック)<br>(LINK) | | MACのリロケータブル版. マイクロソフト形式のリロケータブル・オブジェクトプログラムを出力 |
| | MACRO-80 (マクロエイティー)<br>(LINK-80) | マイクロソフト | 8080, Z-80共用リロケータブル・マクロアセンブラ<br>マイクロソフト形式のリロケータブル・オブジェクトを出力<br>(HEX形式も出力可能) |
| デバッガ | DDT (ディーディーティー) | デジタルリサーチ | 8080デバッガ. CP／M のシステムディスクに標準装備 |
| | SID (シッド) | | DDTのシンボリック・デバッガ版 |
| | ZSID (ゼットシッド) | | SIDのZ-80版 |

図5-1-5　CP/Mベースの代表的ツール一覧表

ここに挙げたものは80系のアセンブラの代表例にすぎず, CP／Mベースの開発ツールは多種多様です. 例えばアセンブラの中には, クロス・アセンブラと呼ばれCP／M上で(つまりZ-80や8080などのマシン上で), 6809, 6502などの他の系統のCPUや, 8086, 68000などの16ビットCPUのソース・プログラムをアセンブルするツールも各種そろっています. また, Z-80や, 8080用のソース・プログラムなどを, 16ビットの8086用のソース・プログラムなどに自動変換する, ソース・プログラム・コンバータと呼ばれるツールなども何種類か用意されています.*

では, 上の表に挙げたアセンブラによるソフトウェア開発ツールの概要と実行例を, エディタから順に紹介していきましょう. ただし, 本章の目的はそれぞれのツールの使用法を解説することではなく, 各ツールの概要を紹介することにあります.

* これらを含め, いろいろなツールの実行例が, 前述の「応用CP／M」に載っている.

## ＊エディタ

4．2章でも述べましたが，エディタは，私たちとコンピュータとの接点にあたる重要なツールで，すべてのソフトウェア開発になくてはならないものです．エディタは文章ファイルを作成するためのソフトウェアであり，ワードプロセッサの原形といえます．ただし，エディタで作成する文章ファイルは，アセンブラや各種言語のソースファイルですから，ワードプロセッサのような，倍角文字やアンダーラインや，印刷時の書式指定などの機能は必要ではありません．

ユーザーはソース・プログラムを作成するために，エディタが備えている各種の編集機能を駆使するわけです．そこで次に，エディタにとって必要不可欠な主な機能を，表にまとめてみましょう．ここでは，文章ファイルを新規に作成する機能と，すでに作成されている文章ファイルの内容を変更する機能があることは基本的な前提です．また，ファイルの内容を変更する場合は，もとのファイルは自動的にバックアップファイルとして保存されるのが普通です．

| 機　能 | そ の 内 容 |
|---|---|
| 読込み | ディスク上にある編集しようとするテキストファイルや，別の任意のテキストファイルを編集エリア*に読み込む |
| 書出し | 編集エリアの任意の部分を，任意のファイル名でディスクにセーブする |
| 挿　入 | 編集エリアの任意の部分へのテキストの挿入（キーボード入力） |
| 削　除 | 編集エリアの任意の部分の削除 |
| 検　索 | 編集対象テキスト全体に対する文字列のサーチ |
| 置換え | 上記のサーチ機能＋その文字列を他の文字列へ置き換える機能 |
| 移　動 | 編集エリアの任意の部分を，任意の箇所へ移動する（もとの部分は削除される） |
| コピー | 編集エリアの任意の部分を，任意の箇所へコピーする |
| バックアップファイル作成 | 編集作業の終了時に，もとのテキストファイルを，バックアップファイルとしてディスク上に残す機能 |

＊編集作業を行うためのメモリ上の作業エリア

図5-1-6　エディタが備えているべき基本的な機能

エディタには，大きく分けてポインタ形式とスクリーンエディタ形式との2種類があります．

ポインタ形式とは，CP／Mに標準装備のエディタ「ED」に代表されるもので，CRTディスプレイが普及していなかった頃(CRTディスプレイの代わりに，プリンタを使っていた頃)はすべてこのタイプでした．この形式のエディタでは，キャラクタ・ポインタ(CPと略して呼ばれる)と呼ぶ目に見えない「カーソル」で編集点*を決定し，各種の編集作業を行います．この形式にはスクリーンエディタ形式にはない特長もあるのですが，やはり編集点が目に見えないので作業能率が悪くなります．最近は強力なスクリーンエディタが各社から発売されているので，純粋なポインタ形式のものは使われなくなってきました．

一方，スクリーンエディタでは，CRTディスプレイ上に表示された文章ファイルを目で確認しながら編集点を決定し，カーソルを移動して各種の編集作業を進めることができます．

しかも，本格的なスクリーンエディタは，カーソルによって直接スクリーンの文章を操作するだけでなく，文字列サーチや，文字列の置換え機能などの，コマンド形式による編集機能をも備えています．

ところが簡易版のスクリーンエディタ(これはBASICマシンのBASICに内蔵されているスクリーンエディタがよい例)は，そのほとんどのものが，コマンド形式による編集機能を持っていません．つまり，文字列サーチや，置換え，移動，コピーなどの機能がないのです．これは，エディタとしては致命的な弱点であり，本格的な開発ツールとしては使いものになりません．もっと機能アップしてほしいものです．

では，先の表に挙げた2つのエディタを使って，それぞれ実際にソース・プログラムをエディットしてみましょう．エディットするソース・プログラムは，これまでの章で例題としてきたメッセージ表示プログラム「MSGOUT．ASM」です．

*変更，削除，追加，置換えなどの編集作業の対象となる位置を指す点．

## ポインタ形式のエディタ「ED」

CP/Mのシステムディスクに標準装備されているエディタ「ED」は,「ED.COM」というファイル名でディスク上に存在しています.*

「ED」の実行例は，4.2章の図4-2-3，および図4-2-7にも示してありますので，ここでは，文字列サーチと置換えの機能などを実行してみましょう．作成または変更しようとするソース・プログラムのファイル名を，「MSGOUT.ASM」とすると，次のコマンドにより「ED」を起動して，エディタの世界に入ることができます.

```
ED MSGOUT.ASM ↲
```

では,「ED」の起動から終了までの，一連の実行例を示します.

図5-1-7　ポインタ形式のエディタ「ED」の実行例

（　）内は使用している各種の編集機能のコマンドの種類を示しています

```
A>ED MSGOUT.ASM ↲ ……既存のファイル「MSGOUT.ASM」に対してエディタを起動する

   : *0a ↲ ……そのファイルの内容をメモリの編集用バッファに読み込む(aコマンド)
  1: *mfEQU^Z0tt ……文字列「EQU」が含まれる行をすべてタイプアウトする(mfコマンド)
  6:  BDOS   EQU   0005H   ; system call entry point |
  7:  CR     EQU   0DH     ; Carriage Return code    | これだけあった
  8:  LF     EQU   0AH     ; Line Feed code          |
  9:  EOS    EQU   00      ; End Of String code      |
BREAK "#" AT ^Z ……もう「EQU」が見つからないという表示
  9: *b ↲ ……キャラクタ・ポインタ(CP)をテキスト(編集を受けるファイル)の先頭にセットする(bコマンド)
  1: *mf:^Z0tt ……「:」が含まれるすべての行をタイプアウトする
 14:  LOOP:                ;          HL=character pointer | これだけ
 26:  CHROUT:                                               | あった
 34:  MESG:   DB     CR,LF,'Good Morning',CR,LF,EOS         |
BREAK "#" AT ^Z ……もう「:」が見つからないという表示
 34: *b ↲ ……CPを先頭にセットする
  1: *msLF^ZLFEED^Z0tt ……すべての文字列「LF」を「LFEED」に書き換え，その行をタイプアウトする(msコマンド)
  8:  LFEED  EQU    0AH     ; Line Feed code                 | 2つの行(ラインNo.
 34:  MESG:  DB     CR,LFEED,'Good Morning',CR,LF,EOS        | 8と34)に計3カ所
 34:  MESG:  DB     CR,LFEED,'Good Morning',CR,LFEED,EOS     | の「LF」があり,
                                                             | 「LFEED」に書き換えられている
BREAK "#" AT ^Z ……もう「LF」が見つからないという表示
 34: *b ↲ ……CPを先頭にセット
  1: *f1 chara^Z0lt ……文字列「1 chara」を捜し，その行の先頭にCPをセットする(fコマンド)
 24:  ;------ 1 character out subroutine ------ ……ラインNo.24にあった
 24: *8t ↲ ……この行から8行分をタイプアウトする(tコマンド)
```

*4.2章の図4-2-1「CP/Mのシステムディスクに含まれる各種のプログラム」を参照.

```
24:  ;------ 1 character out subroutine ------
25:  ;
26:  CHROUT:
27:          MVI     C,2     ; CP/M system call
28:          MOV     E,A     ;          console out
29:          CALL    BDOS    ;                function
30:          RET
31:  ;
24: *8x8k     ……この8行をいったんディスクにセーブし，編集バッファから削除する(xコマンドとkコマンド)
24: *0 ↵      ……現在CPのある行をタイプアウトする(0 ltコマンドの省略形)
24:  ;------ string data area for message ------
24: * ↵       ……CPを次の行に進め，その行をタイプアウトする(1ltコマンドの省略形)
25:  ;
25: * ↵       ……同上
26:  MESG:   DB      CR,LFEED,'Good Morning',CR,LFEED,EOS
26: * ↵       ……同上
27:  ;
27: * ↵       ……同上
28:          END             ; list end
28: *r ↵      ……いったんディスクにセーブした8行のテキストを読み出してCP以後に挿入する(rコマンド)
36: *b ↵      ……CPをテキストの先頭にセットする
 1: *#t ↵     ……テキスト全部をタイプアウトする
 1:  ;---------------------------;
 2:  ;      MESSAGE OUT PROGRAM   ;
 3:  ;   for 8080,8085 ASSEMBLER  ;
 4:  ;---------------------------;
 5:  ;
 6:  BDOS    EQU     0005H   ; system call entry point
 7:  CR      EQU     0DH     ; Carriage Return code
 8:  [LFEED] EQU     0AH     ; Line Feed code
 9:  EOS     EQU     00      ; End Of String code
10:  ;
11:          ORG     100H    ; start address = 100H
                 .
                 .
                 .
21:          INX     H       ; pointer goes up
22:          JMP     LOOP    ; jump for next character
23:  ;
24:  ;------ string data area for message ------
25:  ;
26:  MESG:   DB      CR,[LFEED],'Good Morning',CR,[LFEED],EOS
27:  ;
28:  ;------ 1 character out subroutine ------
29:  ;
30:  CHROUT:
31:          MVI     C,2     ; CP/M system call
32:          MOV     E,A     ;          console out
33:          CALL    BDOS    ;                function
34:          RET
35:  ;
36:          END             ; list end
 1: *e ↵      ……テキストをディスクにセーブしてエディタを終了する(eコマンド)

A>
```

ラインNo.24～31の8行がタイプアウトされている

□部分が変更された箇所

……ブロックで移動された

ここで行った作業は，ラインフィードを表すシンボル名「LF」を「LFEED」に変更したこと（オブジェクト・プログラムには何ら影響しない）と，1文字出力サブルーチンと，メッセージの文字列エリアの位置を入れ換えたこと（オブジェクト・プログラムに影響する）です．このソース・プログラムは，次項の「アセンブラおよびローダ」でアセンブルしてみましょう．

### スクリーンエディタ「Word Master」

Word Masterのマニュアルとディスク

8ビットのCP/M上では，マイクロプロ社のWord Masterが，スクリーンエディタの代名詞となるほど広く使われています．Word Masterは，スクリーンエディタの機能とともに，CP/Mのエディタ「ED」とほぼ同じ内容で，同じコマンド形式の機能（コマンド・コンパチブルと呼ぶ）も備えています．つまり，

**Word Master＝スクリーンエディタ機能＋CP/M「ED」機能**

であるといえばよいでしょう．

ではWord Masterを使って，同じソース・プログラムを新規に作成してみましょう．その過程で，スクリーンエディタの機能や「ED」とコマンド・コンパチブルな機能などをいくつか実行してみます．

図5-1-8　スクリーンエディタ「Word Master」の実行例

```
A>WM MSGOUT.ASM ↵
```
……Word Masterを起動して，新しいファイル「MSGOUT.ASM」を作成する

```
MicroPro WORDMASTER release 1.07J MPJ ID # NE4G-020
 COPYRIGHT (C) 1978  MICROPRO INTERNATIONAL CORPORATION

NEW FILE
```

Word Masterが起動すると，1〜2秒このオープニング・メッセージが表示された後，すぐに画面がクリアされ，スクリーンエディタのモードに入る

```
;---------------------------;
;     MESSAGE OUT PROGRAM    ;
;   for 8080,8085 ASSEMBLER  ;
;---------------------------;
;
BDOS    EQU     0005H    ; system call entry point
CR      EQU     0DH      ; Carriage Return code
LF      EQU     0AH      ; Line Feed code
EOS     EQU     00       ; End Of String code
;
        ORG     100H     ; start address = 100H
;
        LXI     H,MESG   ; get top address of message
LOOP:                    ;         HL=character pointer
        MOV     A,M      ; get character code to output
        ORA     A        ; end of string? (A=00?)
        RZ               ; if 'Z'=1, end of this program
        PU                 character pointer
                             ut
                                pointer
```

キー入力したものは，そのままテキストの入力となる．タブキーを使って各フィールドをそろえ，ソース・プログラムを入力していく

```
;---------------------------;
;     MESSAGE OUT PROGRAM    ;(上のリストの一部の書き換えを行う)
;---------------------------;
;
BDOS    EQU     0005H    ; system call entry point
CR      EQU     0DH
LF      EQU     0AH
EOS     EQU     00       ; End Of String code
;
        CSEG
;
        LXI     H,MESG   ; get top address of message
NEXT:                    ;         HL=character pointer
        MOV     A,M      ; get character code to output
        ORA     A        ; end of string? (A=00?)
        RZ               ; if 'Z'=1, end of this program
        PUSH    H        ; save character pointer
                           character out
                              haracter pointer
```

CSEG……カーソルを「ORG」のOの箇所に移動し，「CSEG」と入力した後，ctrl-Kを入力する．カーソルの右側の行が全部削除され，このように書き換えられる

NEXT:……カーソルを「LOOP」のLに移動して「NEXT」と入力すると，このように書き換えられる

```
            VIDEO MODE SUMMARY  (TYPE ^J FOR NEXT FRAME)

    ^O   INSERTION ON/OFF               RUB   DELETE CHR LEFT
    ^S   CURSOR LEFT CHAR               ^G    DELETE CHR RIGHT
    ^D   CURSOR RIGHT CHAR              ^¥    DELETE WORD LEFT
    ^A   CURSOR LEFT WORD               ^T    DELETE WORD RIGHT
    ^F   CURSOR RIGHT WORD              ^U    DELETE LINE LEFT
    ^Q   CURSOR RIGHT TAB               ^K    DELETE LINE RIGHT
    ^E   CURSOR UP LINE                 ^Y    DELETE WHOLE LINE
    ^X                                  ^I    PUT TAB IN FILE
                                              PUT CRLF IN FILE
                                              NEXT CHR 4X
```

スクリーンエディタのモードでは，いつでもctrl-Jの入力により，
このヘルプメニューが表示される．このメニューは全部で4ページある

ヘルプメニューでctrl-J以外のキー入力をすると，もとのテキストが表示される

```
           LXI      H,MESG    ; get top address of message
   NEXT:                      ;             HL=character pointer
           MOV      A,M       ; get character code to output
           ORA      A         ; end of string? (A=00?)
           RZ                 ; if 'Z'=1, end of this program
           PUSH     H         ; save character pointer
           CALL     CHROUT    ; character out
           POP      H         ; restore character pointer
           INX      H         ; pointer goes up
           JMP      NEXT      ; jump for next character
 ;[ESC]……ここでESCキーの入力により，ポインタ形式のエディタ・モード(コマンドモード)に移る．コマンドモードのプロンプト「*」が表示される
 *B ↵ ……キャラクタ・ポインタ(CP)をテキストの先頭にセットする．(bコマンド)
 *<FMOV$0TT> ↵ ……文字列「MOV」を含むすべての行をタイプアウトする(fコマンドとtコマンド)
           MOV      A,M       ; get character code for output   | この2つがあった
           MOV      E,A       ;          console out            |
  ## FMOV$0TT> ……もう「MOV」は見つからないという表示
 *B ↵ ……CPをテキストの先頭にセット
 *<SCR$CRET$0TT> ↵ ……文字列「CR」をすべて「CRET」に書き換え，
 CRET      EQU      0DH   その行をタイプアウトする(sコマンドとtコマンド)
 MESG:     DB       CRET,LF,'Good Morning',CR,LF,EOS     | この2行の3箇所が「CRET」
 MESG:     DB       CRET,LF,'Good Morning',CRET,LF,EOS   | に書き換わった
  ## SCR$CRET$0TT> ……もう「CR」は見つからないという表示
 *V ↵ ……コマンドモードを終了し，スクリーン・エディタのモードに移る(vコマンド)
```

（左側縦書き：ポインタ形式のエディタモード）

再びスクリーンエディタのモードに移った．カーソルによるエディットが可能

```
        CALL    BDOS    ;                    function
        RET
;
;------ string data area for message ------
;
MESG:   DB      CRET,LF,'Good Morning',CRET,LF,EOS
;
        END             ; list end


[ESC]……ESCキーの入力により，もう一度コマンドモードに移る
*E ↵ ……コマンドモードから，e(END)コマンドにより，テキストをディスクにセーブして，当エディタを終了する

A>
```

ここで作成されたものはシンボル名が一部異なりますが，今までの例題のソース・プログラムと同じ内容です．このソース・プログラムも，次項でアセンブルしてみましょう．

### ＊アセンブラおよびローダ

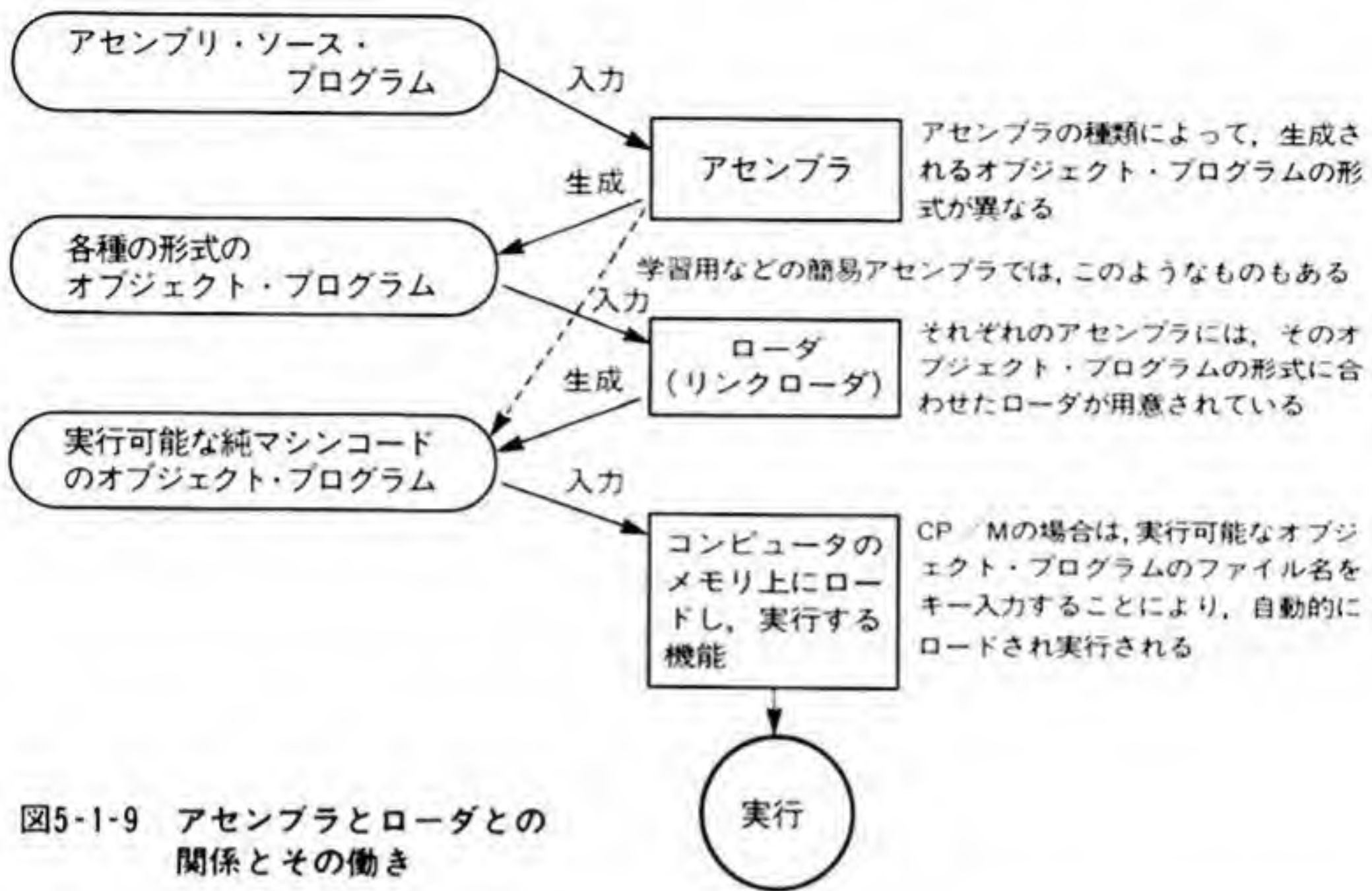
アセンブラとローダは対で提供されます．つまり，アセンブラから出力されるオブジェクト・プログラムの形式に合わせたローダが必要なわけです．4章でも簡単に述べましたが本格的なアセンブラから出力されるオブジェクト・プログラムの形式は，そのままで実行が可能な純マシンコードではありません．純マシンコードを出力させるのは，アセンブラにとって最も簡単なことなのですが，多くの場合，アセンブル作業の後には，生成されたオブジェクト・プログラムをメモリ上にロードして実行するだけの単純な作業だけではなく，さらにいろいろな作業が続きます．

例えば，メモリ上のロード・アドレスを意識することなく，正しいアドレスにオブジェクト・プログラムをロードしたり，他のマシンとの間でオブジェクト・プログラムの交換をしたり，別々に開発した複数のオブジェクト・プログラムを結合して，任意のロード・アドレスをもつ1本のオブジェクト・プログラムを作成するなどのためには，純マシン語では対応不可能であるか，もしくは非常に困難です．そこで，これらの作業を容易に実現する機能を持った，特別な形式のオブジェクト・プログラムが必要なのです．

これらのことから，アセンブラには，そてぞれのアセンブラが出力するオブジェクト・プログラムの形式に合ったローダが必要となるわけです．

4章でも述べましたが，この特別な形式には「インテル HEX 形式」と，「マイクロソフト社リロケータブル・オブジェクト形式」の2つが国際的な標準となっています．これらについては，それぞれ APPENDIX　2および11章を参照してください．

ここで，アセンブラとローダとの関係とその働きを図で示しておきますので，再度確認しておいてください．

図5-1-9 アセンブラとローダとの関係とその働き

さて，アセンブラには，基本的には次の3つのタイプがあり，その組合せによるものを含めて，主に4種類が使われています．

- **アブソリュート・アセンブラ**
- **リロケータブル・アセンブラ**
- **マクロアセンブラ**

（リロケータブル・アセンブラ，マクロアセンブラ）⇨ **• リロケータブル・マクロアセンブラ**

この中で本格的なアセンブラとして最も多く使われているのが，後の2つを組み合わせた「リロケータブル・マクロ・アセンブラ」と呼ばれるものです．ではまず予備知識としてそれぞれのアセンブラの概略を解説しておきましょう．

### アブソリュート・アセンブラ「ASM」

4．2章で実行例を示したCP/M上のアセンブラ「ASM」(8080アセンブラ)がアブソリュート・アセンブラの代表例です．

このタイプのアセンブラは，アセンブルにより生成されたオブジェクト・プログラムがロードされるべきメモリ上のアドレスが，ソース・プログラム中のORG擬似命令で宣言され，絶対的(アブソリュート)に定まります．つまりできあがったオブジェクト・プログラムは，ソース・プログラムのORG擬似命令で指定したメモリ・アドレス上でのみ動作が可能です．

例えば4章の実行例で作成されたオブジェクト・プログラムは，アドレス$0100_H$からメモリ上にロードし，このアドレスから実行を開始するプログラムです．もしこれを1番地でもずらしてロードすると，プログラムは動作しません．ロード・アドレスを変更したい場合には，まずソース・プログラムのORG擬似命令のアドレス指定を変更してから再アセンブルしなければなりません．

このタイプのアセンブラの多くは，出力するオブジェクト・プログラムの形式としてインテルHEX形式を採用しています．この形式のオブジェクト・プログラムは，それ自身がロードされるべきメモリのアドレス情報を持っており、外部からロード・アドレスを指定することなくメモリ上の正しい位置にロードすることができます．

4章の**図4－2－11**に実行例を示したように、CP／Mのローダ「LOAD」で，このインテルHEX形式のオブジェクトファイルを実行可能な純マシン語ファイルに変換できます．「ASM」と「LOAD」の実行例は，ここでは示しませんが，必要であれば4．2章をご覧ください．

**リロケータブル・アセンブラ**

リロケータブル・アセンブラとは，前項のCP／Mのアセンブラ「ASM」に代表される，アブソリュート・アセンブラに対するもので，任意のメモリ・アドレスにロード可能な形式のオブジェクト・プログラムを出力します．しかしリロケータブルだけの機能を持ったアセンブラで一般的に広く使われているものはなく，ほとんどのものは次項のマクロ機能と組み合わせて，リロケータブル・マクロ・アセンブラという形式を採っています．

このリロケータブル・アセンブラによって生成されたオブジェクト・プログラムは，メモリ上へロードする際の特定のロード・アドレスが規定されていません．最終的には，このオブジェクト・プログラムがローダ(この場合の

ローダを「リンクローダ」と呼ぶ)によって，純マシンコードに変換される時点で，任意のロード・アドレスを持ったオブジェクト・プログラムが生成されます．

このような「ロード・アドレス後決め形式」のオブジェクト・プログラムは，リロケータブル形式のオブジェクト・プログラム，つまりリロケータブル・オブジェクト・プログラムと呼ばれており，"再配置可能目的プログラム"と直訳されているものがそれにあたります．

また，たいていのリロケータブル・アセンブラによるオブジェクト・プログラムは，次に述べるもうひとつの機能を持っています．大きなプログラムの開発になると，ソース・プログラムをいくつかに分割して(その一つひとつをモジュールと呼ぶ)別々に開発することになります．これらを，リロケータブル・アセンブラで別々にアセンブルすることによって複数のリロケータブル・オブジェクト・プログラムができます．これらのオブジェクト・プログラムはリロケータブル形式ですので，互いに結合(リンク)して，1本のオブジェクト・プログラムにまとめることができます.*

このような複数のリロケータブル・オブジェクト・プログラムをリンクして，1本の純マシン語やインテルHEX形式などのオブジェクト・プログラムに変換するのが，「リンクローダ」と呼ばれるローダです．

以下に，これらの機能の概念を図で示しておきます．なお，リロケータブル・アセンブラと次項のマクロ・アセンブラについては，別に章を設けて解説します．詳しくは，11章「リロケータブル・マクロアセンブラの概念と使い方」を参照してください．

＊このリンク機能では，それぞれのモジュールで共通のシンボルやラベルを使うことを考慮している．ただ単純にオブジェクトをつなぎ合わせるだけではない．

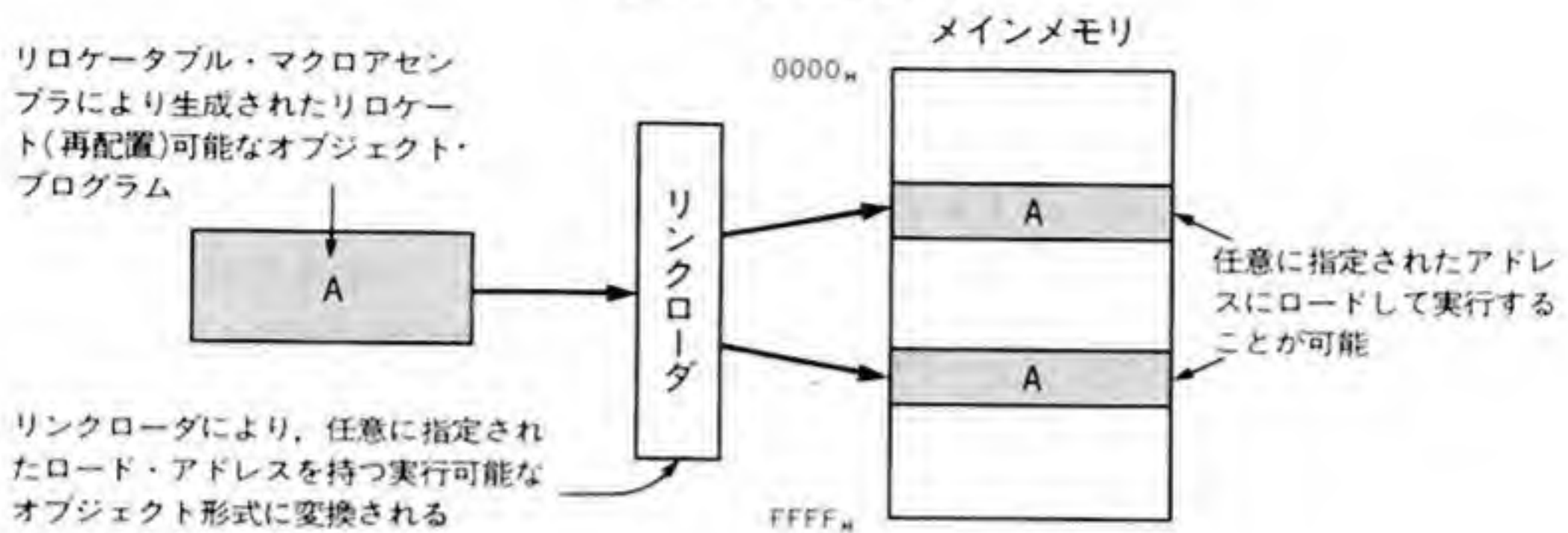
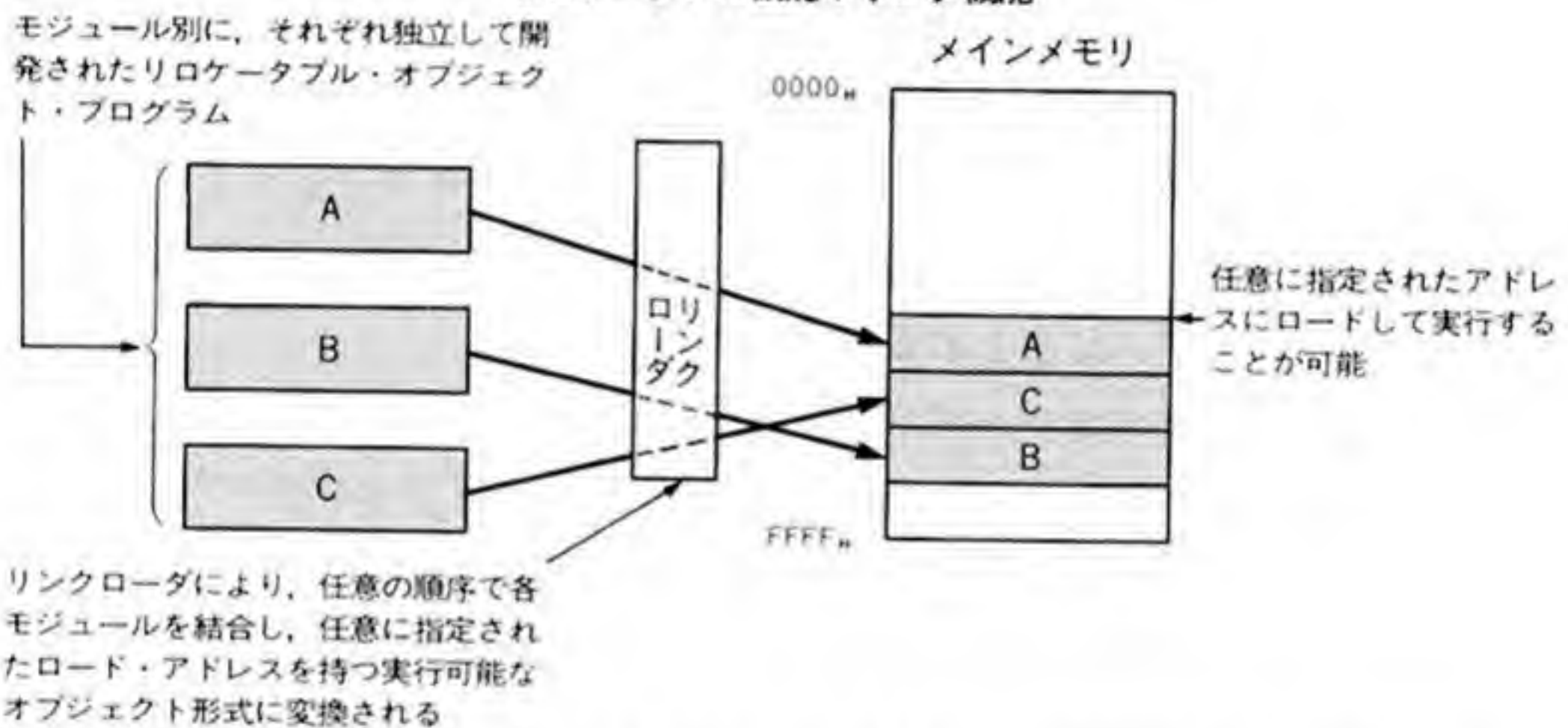



図5-1-10　リロケータブル・アセンブラによるオブジェクト・プログラムの機能

## マクロアセンブラ「MAC」

マクロアセンブラMACのマニュアル

マクロ機能だけを持ったアセンブラの代表は，デジタルリサーチ社の「MAC」(Z-80, 8085, 8080CPU用アセンブラ)です．マクロアセンブラも，マクロ機能を使わなければ普通のアセンブラと同じです．

マクロ機能とは，ユーザーがある機能を持つプログラムのブロック(一見，サブルーチンのようであるが概念が異なり多くの機能がある)を作成して，それに「名前」(マクロ名と呼ぶ)をつけて定義しておき(マクロ定義と呼ぶ)，その後ソース・プログラム中でこのマクロ名をあたかもCPU命令のように使うことができる機能です．このソース・プログラムをマクロアセンブラでアセンブルすることにより，マクロ名を使った部分が，その名前で定義されているブロックのプログラムに展開されるわけです．

この「MAC」と，そのリロケータブル・マクロアセンブラ版「RMAC」のマクロ機能は非常に高度な応用が可能であり，注目に値します．マクロアセンブラの概念はそれほど難しいものではありませんが，アセンブラの基礎を全体的に理解してから学ぶ方がよいでしょう．

### リロケータブル・マクロアセンブラ「M 80」,「RMAC」

リロケータブル・マクロアセンブラM80のマニュアルとディスク

みなさんが，アセンブラによるソフトウェア開発を本格的に行うようになった場合に使うことになるであろうアセンブラは，リロケータブル・アセンブラと，マクロアセンブラの機能を合わせた「リロケータブル・マクロアセンブラ」でしょう．

リロケータブル・マクロアセンブラの代表的なものはマイクロソフト社の「M80」(Z-80, 8085, 8080CPU用アセンブラ)ですが，デジタルリサーチ社からもマクロアセンブラ「MAC」のリロケータブル版である「RMAC」が発売されています．この2つは，ほぼ同じ機能を持ち，同一形式(マイクロソフト社リロケータブル・オブジェクト形式)のオブジェクト・プログラムを出力します．従って互いのオブジェクト・プログラムには互換性があり，両者を混在させてもリンクが可能です．

とはいっても，もちろん両社ともオリジナルのリンクローダを持っていて，「M80」(MACRO-80)には「L80」(LINK-80)というリンクローダが組み合わされ，「RMAC」(Relocatable MACro assembler)には「LK80」(これもLINK-80と呼ぶ)というリンクローダが組み合わされています．

図5-1-11 リロケータブル・マクロアセンブラ「M80」の実行例

```
A>REN MSGOUT.MAC=MSGOUT.ASM ……M80アセンブラのソース・プログラム名は「××××.MAC」でなければならないので，ファイル名を「MSGOUT.MAC」に変更する

A>M80 MSGOUT,MSGOUT=MSGOUT ……ソース・プログラム「MSGOUT.MAC」をアセンブルする

No Fatal error(s)
アセンブル・エラーなく，アセンブル終了

A>DIR MSGOUT.* ↵ ……ファイル「MSGOUT」に関するすべてのファイル名をタイプアウトする

A: MSGOUT    MAC : MSGOUT    PRN : MSGOUT    REL
   ソース・プログラム  生成されたアセンブル・  生成されたリロケータブル・
                     リスト・ファイル      オブジェクト・プログラム

A>L80 /P:4000,MSGOUT,MSGOUT/N/X/E ↵ ……ロード・アドレスを持っていないリロケータブル・オブジェクト・プログラム「MSGOUT.REL」に対して，ロード・アドレス4000Hを指定してリンクローダを実行，インテルHEX形式のオブジェクトを得る

Link-80  3.44  09-Dec-81  Copyright (c) 1981 Microsoft

Data    4000    4027    <   39>

40979 Bytes Free
[0000   4027        64]
リンクローダの実行終了

A>DIR MSGOUT.* ↵ ……すべての「MSGOUT」のファイル名をタイプアウトする

A: MSGOUT    MAC : MSGOUT    PRN : MSGOUT    REL : MSGOUT    HEX
                                           リンクローダの実行により生成されたインテルHEX形式のオブジェクト・プログラム

A>TYPE MSGOUT.HEX ↵ ……生成された4000HスタートのインテルHEX形式のオブジェクト・プログラムをタイプアウトする

:2040000002116407EB7C8E5CD0F40E123C303400E025FCD0500C90D0A476F6F64 → 204D6F7229
:074020006E696E670D0A00D6
:00000001FF

A>
   ロード・アドレス     チェックサム     オブジェクト・プログラム     チェックサム
```

リロケータブル・マクロアセンブラについては11章で詳しく解説していますので，ここでは「M80」と「L80」の簡単な実行例だけを示しておきましょう．本章のエディタの項でWord Masterによって作成された，おなじみのメッセージ表示プログラムのソース・プログラムをアセンブルします．

「M80」は，Z-80用のザイログ形式ニーモニック，および8080用のインテル形式ニーモニックの両方のソース・プログラムに対してアセンブルが可能です．「M80」を実行する際にここでの実行例のように，Z-80か8080かの指定をしない場合には，8080用としてアセンブルが行われます．Z-80用としての実行例は次の章で示します．

なお，ここでのソース・プログラムでは，今までの「ORG 100H」の行を「CSEG」に書き換えている点に注意してください．

## *デバッガ

Z-80用の代表的なデバッガは，デジタルリサーチ社の「ZSID」(Z-80 Symbolic Instruction Debugger)です．これは，CP/Mのシステムディスクに標準装備されている8080用デバッガ「DDT」(Dynamic Debugging Tool)のZ-80版であり，さらにいくつかの機能が拡張されています．

デバッグ作業については4章でも少し触れましたが，開発過程の中で特にやっかいな作業です．できあがったオブジェクト・プログラムは，必ずといってよいほどバグがあります．そこでこのオブジェクト・プログラムをデバッガを通して実行し，デバッガの機能を駆使してバグを捜し出すわけです．デバッグにはアセンブラの知識はもちろん，コンピュータのハード，ソフトを問わずありとあらゆる知識を総合して立ち向かう必要があります．

デバッガについては，次の「DDT」を含めて10章で詳しく解説しますので，ここではデバッグツールの簡単な紹介にとどめておきます．

### 8080デバッガ「DDT」

「DDT」はCP/Mのシステムディスクに含まれている8080用デバッガで，古くから多くの人に愛用され，その後に作られたデバッガのコマンド形式などに大きな影響を与えています．例えば次節で紹介する「DUAD」というスタンドアローンの開発ツールに含まれるデバッガやPC-8801のモニタなども，

DDTにたいへんよく似たコマンド形式を採用しています．

ここでは前項の「M80」と「L80」の作業で作成された，例題プログラムのオブジェクト・プログラム（$4000_H$を実行開始アドレスとするインテルHEX形式のもの）をメモリ上にロードして，「DDT」のいくつかの機能を実行してみましょう．

図5-1-12　CP/Mデバッガ「DDT」の実行例

```
A>DDT MSGOUT.HEX ↵ ……DDTを起動して，インテルHEX形式のオブジェクト・プログラム「MSGOUT.HEX」を実行可能な純マシン
                        語のオブジェクトに変換して，「MSGOUT.HEX」自身が持っているロード・アドレス上にロードする
DDT VERS 2.2 ……………DDTが起動した
NEXT  PC
4027 0000    ……………「MSGOUT.HEX」のロード完了

-D3FF0 403F ↵ ……………アドレス3FF0H～403FHのメモリ内容をダンプする
3FF0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
4000 21 16 40 7E B7 C8 E5 CD 0F 40 E1 23 C3 03 40 0E !.@~.....@.#..@.
4010 02 5F CD 05 00 C9 0D 0A 47 6F 6F 64 20 4D 6F 72 ._......Good Mor
4020 6E 69 6E 67 0D 0A 00 00 00 00 00 00 00 00 00 00 ning............
4030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
                                                     左の16進表示に対応するアス
                                                     キー表示部
-L4000 ↵ ……………アドレス4000Hからのメモリ内容を逆アセンブルする
  4000  LXI   H,4016
  4003  MOV   A.M          実行可能なオブジェクト形式に変換され，
  4004  ORA   A            4000Hからロードされている例題プログラム
  4005  RZ
  4006  PUSH  H
  4007  CALL  400F     逆アセンブルによって表示されたソース・プログラム
  400A  POP   H        逆アセンブルとはオブジェクト・プログラムから，そのソース・プログラムを作り出す機能
  400B  INX   H
  400C  JMP   4003
  400F  MVI   C,02
  4011  MOV   E,A
-A3000 ↵ ……………アドレス3000Hから，ライン・アセンブルによってオブジェクトコードをメモリ上にロードする
3000  CALL 4000  ┐ソース・プログラムを入力，1行ごとにオブジェクトに変換される
3003  RST 7 ↵   ┘
3004  . ↵ ……………ピリオドの入力によりライン・アセンブルを終了

-D3000 3005 ↵ ……………ライン・アセンブルの結果の確認
3000 CD 00 40 FF 00 00 ..@...
                       ……………オブジェクトコードが生成されている
-G3000 ↵ ……………アドレス3000Hから実行，例題プログラムは1つのサブルーチンでもあるので，
                   その4000Hのサブルーチンをここからコールしている
Good Morning    例題プログラムの実行によるメッセージが表示された

*3003 ……………実行が3003H（RST 7，Z-80ではRST 38H）で終わったことを示す

-^C ↵ ……………Ctrl-Cの入力によりDDTを終了する

A>
```

## Z-80シンボリック・インストラクション・デバッガ「ZSID」

ZSIDのマニュアル

「ZSID」は，「DDT」のZ-80版と考えれば，「DDT」と同じコマンドで同じように使うことができますが，さらに高度な機能も追加されています．例えばデバッガの各種のコマンドを実行する際に，ソース・プログラム中のシンボル名をアドレスの代わりに使うことができます．

「DDT」ではアドレス指定にシンボル名は使えませんので，すべて「xxxxH」のように絶対アドレスで指定しなければなりません．ところが大きなプログラムを開発する場合は，長大な1本のソース・プログラムを作成するのではなく，それをいくつかの部分(モジュールと呼ぶ)に分割し，それぞれ独立して開発を行い，最後にリンクローダで1本にまとめる手法をとります．これを「モジュール別ソフト開発法」などと呼びますが，この場合は，ソース・プログラムとオブジェクト・プログラムのアドレスの対応が非常に困難です．そこで，絶対アドレス値を意識せず，シンボル名やラベルを使ってデバッグ可能なデバッガが必要になるわけです．これがシンボリック・インストラクション・デバッガの名の由来です．これについても詳しくは10章で解説し，そこで実行例を示します．

# 5/2 流通OSを使用しないディスク・ベースのツール

CP/Mなどの流通OSを使用せずにそれぞれのマシンに組込みのBASICをベースにする，アセンブラのソフトウェア開発ツールも数多く発売されています．いずれも本格的な開発や大規模な開発には向きませんが，BASIC上で動作するあまり大きくないマシン語プログラムの開発には実用性がある製品もあります．

これらのほとんどは，パーソナル・コンピュータの機種別にそれぞれ専用の製品が用意されており，ある機種用のものを，他の機種では実行できない場合が多いようです．この点，流通OSをベースにしたものであれば，マシンのメーカーや機種にかかわらず同一の流通OS上であれば，ほとんどのもので実行可能であり，ソース・プログラムやオブジェクト・プログラムの互換も容易です．*

このような流通OSをベースにしないスタンドアローンの開発ツールは，学習用からある程度実用可能なものまで様々ですが，ここではその中からアスキー製の「DUAD」を紹介しましょう．

＊ただしスクリーンエディタに関しては，それぞれのCP/Mマシンのスクリーン表示部のコントロール法が異なる場合がある．その際は用意されている変更プログラムを使って各機種に合わせる作業が必要．

# DUAD

**DUADのマニュアルとディスク**

DUAD(Disk Utilities Assembler dis-assembler and Debugger)にはPC-8001, PC-8801用などがありますが，ここでは「DUAD-88D」というPC-8801用のものを使います．

DUADは，次に示すようにZ-80アセンブラによるソフトウェア開発に必要な各種のツールで構成されている総合開発ツールです．

- **スクリーンエディタ**
- **アセンブラ**　(インテルHEX形式のオブジェクト・プログラムを出力するアブソリュート・アセンブラ)
- **ディスアセンブラ**　(ラベルつきのソース・プログラムを出力する逆アセンブラ)*
- **ローダ**　(インテルHEX形式のオブジェクト・プログラムを，実行可能な純マシン語のオブジェクト・プログラムに変換して，メモリへロードする)
- **デバッガ**　(CP／Mのデバッガ「DDT」と同レベル機能を持ったデバッガ)

DUADについてあまり詳細に説明する余裕がありませんが，ここで実際に

*逆アセンブラについては10.1章を参照．

「DUAD-88D」を使って，一通りの開発を行ってみましょう．

例題とするプログラムは，今までと同様のメッセージ表示プログラムです．今度はCP/M上ではなく $N_{88}$-BASIC上で実行するプログラムを作成しますので，1文字出力サブルーチン「CHROUT」と実行終了時の処理がCP/Mの場合と異なりますので注意してください．ソース・プログラムは，**図5-2-1**のアセンブルリストを参照してください．

**図5-2-1 DUADによるソフト開発の一連の実行例**

DUADのディスクをセットしてリセット・ボタンを押すと，自動的に起動が行われる

```
*** D U A D - 8 8 D   Ver 1.0 ***

        Serial No. 330547

         Command : Utility
               1 : FILES 2
               2 : Screen Editor
               3 : Assembler
               4 : Dis-assembler
               5 : Loader
               6 : debugger DD-8
               7 : debugger DD-8'
               8 : Quit

        what's command ?
```

DUADが起動するとこのようなメニューが表示され，何を行うかを選択するためのキー入力待ちとなる
ここではすでにメニュー2:のスクリーンエディタを使って，「ソース・プログラム「MSGOUT.e」が作成されているとする(ソース・プログラムは，アセンブルリストを参照)

メニュー3:のアセンブラを選択すると，Z-80アセンブラが起動する

```
+++ ASSEMBLER Ver 1.0 +++

Source file name? 2:MSGOUT.e
PASS-1
PASS-2

D027                 END                      ; list end

OPTION is  'l p o f d s r h n' and drive number
```

入力するソース・プログラム名を聞いてくるので，ドライブ2:にある「MSGOUT.e」を指定した

アセンブル処理終了

生成されるオブジェクト・プログラムや，アセンブルリストなどの処理を聞いてくる

```
Option? of2 ↵ ……………オブジェクト・プログラムとアセンブル・リスト・ファイルを，
                      ドライブ2:のディスクにセーブするように指示する
PASS-2


CREATE   2:MSGOUT.hex
                        これらを生成して，ドライブ2:にセーブしたことを示す表示
CREATE   2:MSGOUT.lst

D027               END                    ; list end

今回のアセンブル作業は完了した
OPTION is  'l p o f d s r h n' and drive number

Option? STOP キーの入力によりDUADのメニュー画面に戻る
```

メニュー8:のQuit(終了)コマンドでDUADを終了してBASICに戻り，LLISTで
アセンブル・リスト・ファイル「MSGOUT.lst」をタイプアウトする

```
                                                        ソース・プログラム
           ;---------------------------;
           ;     MESSAGE OUT PROGRAM    ;
           ;---------------------------;
           ;
3E0D       PCOUT:  EQU   3E0DH    ; N88-BASIC chrout entory p.
000D       CR:     EQU   0DH      ; Carriage Return code
000A       LF:     EQU   0AH      ; Line Feed code
0000       EOS:    EQU   00       ; End Of String code
           ;
                   ORG   0D000H   ; start address = D000H
           ;
D000 2116D0        LD    HL,MESG  ; get top address of message
D003       LOOP:                  ;        HL=character pointer
D003 7E            LD    A,(HL)   ; get character code to output
D004 B7            OR    A        ; end of string? (A=00?)
D005 CA11D0        JP    Z,EXIT   ; if 'Z'=1, end of this program
D008 E5            PUSH  HL       ; save character pointer
D009 CD12D0        CALL  CHROUT   ; character out
D00C E1            POP   HL       ; restore character pointer
D00D 23            INC   HL       ; pointer goes up
D00E C303D0        JP    LOOP     ; jump for next character
           ;
D011       EXIT:
D011 FF            RST   38H      ; exit this program
           ;
           ;------ 1 character out subroutine ------
           ;
D012       CHROUT:
D012 CD0D3E        CALL  PCOUT    ; N88-BASIC chrout entory p.
D015 C9            RET
```

DUADの場合は，EQUとも一部の［:］が必要

1文字出力ROM内ユールのエントリー・ポイント

マシン語プログラムのロード可能なアドレスは，各機種によって異なるので注意

```
                  ;
                  ;------ string data area for message ------
                  ;
D016 0D0A476F MESG:   DB    CR,LF,'Good Morning',CR,LF,EOS
D01A 6F64204D
D01E 6F726E69
D022 6E670D0A
D026 00
                  ;
D027                      END                  : list end
```

□部分がN88-BASIC上で実行するための変更箇所。CP/M上で実行する場合のソース・プログラムと異なるので注意すること

メニュー 5:のローダを選択すると，ローダが起動する

```
+++ LOADER  Ver 1.0 +++

OFFSET ? 0↵ ………この場合，メモリ上へロードするのにオフセットは必要ないので0を入力する
File name? 2:MSGOUT↵ ……入力するインテルHEX形式のオブジェクト・プログラムは「MSGOUT.hex」である
                        「.hex」は入力しない
D000-D026 | 実行可能な純マシン語のオブジェクト・プログラムに変換されて，アドレスD000H～D026Hにロードされた
 D000-D026|

Ok ………自動的にBASICに戻る
MON ………モニタへ入る

h]DD000,D02F↵ ………D000H～D02FHをダンプして，メモリ内容を確認する
D000 21 16 D0 7E B7 CA 11 D0 E5 CD 12 D0 E1 23 C3 03   ! ミ~キハ ヱヘ ミト#テ
D010 D0 FF CD 0D 3E C9 0D 0A 47 6F 6F 64 20 4D 6F 72   ミ ヘ >ノ  Good Mor
D020 6E 69 6E 67 0D 0A 00 0A 47 6F 6F 64 20 4D 6F 72   ning      Good Mor
                               当プログラムのオブジェクト     アスキー表示部
h]GD000↵ ………当プログラムを実行，D000Hスタート
Good Morning | 当プログラムの実行による表示

h]^b ………Ctrl-Bの入力によりモニタを終了
Ok ………BASICに戻った
```

# 5/3 カセット・ベースのツール

各マシンのBASIC上で動作する，カセット・ベースの学習用アセンブラも数多く発売されています．それらはBASICで作られているものもあれば，マシン語で作られているものもありますが，多くのものはBASICが持っているロード，セーブやエディタなどの機能を全面的に利用しています．

これらの学習用アセンブラでは，ソース・プログラムはBASICのエディタ機能を使って，注釈文(行の始めに[']を置く)を利用して作成します．アセンブルを実行して生成されたオブジェクト・プログラム(ほとんどが純マシン語を出力する)は，メモリ上に置かれたままですので，これをBASICのコマンドや内蔵モニタを使ってカセットにセーブするというわけです．

これらのアセンブラはあくまで学習用であり，当然多くの機能を持っているわけではありません．とはいっても中にはよく考えられている製品もあり，アセンブラの基本学習を行うにはたいへん有効でしかも安価です．

ここでは，よく考えられている学習用のアセンブラのひとつであるアスキーの「MF ASM」を紹介しましょう．

# MF ASM

「MF ASM」は，アスキー出版局発行の『PC-8801　マシン語入門』の中で実習用に使われているアセンブラを単独の製品としたものです．

まずこのアセンブラによるソフトウェア開発の手順を図で示します．

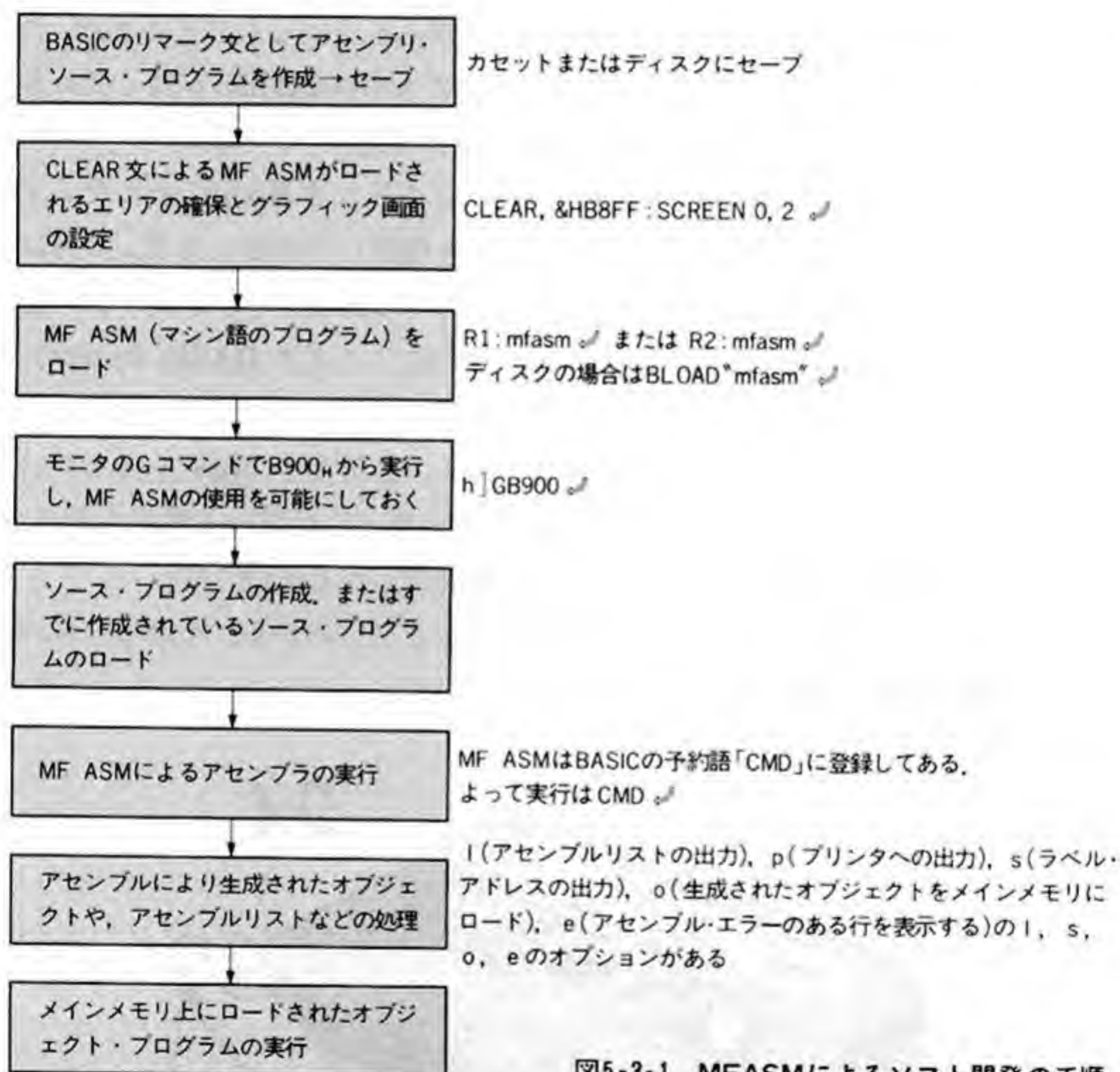



図5-3-1　MFASMによるソフト開発の手順

ではこの「MF ASM」を使って，前項のDUADの場合と同じソース・プログラムを例に，実際に一通りの開発作業を行ってみましょう．

**図5-3-2　MFASMによる一連の実行例**

BASICのリマーク文で書かれたアセンブリ・ソース・プログラム、各行の頭に[']を必ず置くことに注意

```
100 ';---------------------------;
110 ';      MESSAGE OUT PROGRAM    ;
120 ';---------------------------;
130 ';
140 '  PCOUT:     EQU      3E0DH
150 '  CR:        EQU      0DH
160 '  LF:        EQU      0AH
170 '  EOS:       EQU      00
180 ';
190 '             ORG      0D000H
200 ';
210 '             LD       HL,MESG
220 '  LOOP:
230 '             LD       A,(HL)
240 '             OR       A
250 '             JP       Z,EXIT
260 '             PUSH     HL
270 '             CALL     CHROUT
280 '             POP      HL
290 '             INC      HL
300 '             JP       LOOP
310 ';
320 '  EXIT:
330 '             RST      38H
340 ';
350 ';------ 1 character out subroutine ------
360 ';
370 '  CHROUT:
380 '             CALL     PCOUT
390 '             RET
400 ';
410 ';------ string data area for message ------
420 ';
430 '  MESG:      DB       CR,LF,'Good Morning',CR,LF,EOS
440 ';
450 '             END
```

MF ASMの場合にはEQUのシンボルにも〜〜部の[:]が必要

（コメント部分は省略してある）

```
Ok
CLEAR,&HB8FF:SCREEN 0,2 ↵ ………MF ASM自身のマシン語プログラムがロードされるエリアを確保する
Ok
MON ↵ ………モニタに移る

h]R ↵ ………MF ASMをテープからロードする

h]GB900 ↵ ………MF ASMを使用可能にするためのイニシャライズ、B900Hから実行する
h]^b ↵ ………Ctrl-Bを入力してBASICに戻る
Ok
LOAD "MSGOUT ↵ ………あらかじめ作成してテープにセーブしてあったソース・プログラム「MSGOUT.BAS」を
Ok                    ロードする
CMD ↵ ………アセンブラの実行、BASICの予約語「CMD」を実行することにより、MF ASMが実行される

PASS-1                 *END*
D02A

PASS-2                 *END*
D02A      アセンブル終了
```

```
OPTION is " l p s o e " ……生成されたオブジェクト・プログラムや，アセンブルリストの処理を聞いてくる
OPTION ?lpo ↵ ……………… アセンブルリストをプリンタへ出力して，生成されたオブジェクト・プログラムをメインメ
                          モリにロードするように指示する
```

この後，プリンタにアセンブルリストが出力される

プリンタに出力されたアセンブルリスト

```
**    MF-ASSEMBLER(1) (PC-8801)    **

                 ;----------------------------;
                 ;     MESSAGE OUT PROGRAM    ;
                 ;----------------------------;
                 ;
3E0D             PCOUT:  EQU   3E0DH
000D             CR:     EQU   0DH
000A             LF:     EQU   0AH
0000             EOS:    EQU   00
                 ;
                         ORG   0D000H
                 ;
D000 2116D0              LD    HL,MESG
D003             LOOP:
D003 7E                  LD    A,(HL)
D004 B7                  OR    A
D005 CA11D0              JP    Z,EXIT
D008 E5                  PUSH  HL
D009 CD12D0              CALL  CHROUT
D00C E1                  POP   HL
D00D 23                  INC   HL
D00E C303D0              JP    LOOP
                 ;
D011             EXIT:
D011 FF                  RST   38H
                 ;
                 ;------ 1 character out subroutine ------
                 ;
D012             CHROUT:
D012 CD0D3E              CALL  PCOUT
D015 C9                  RET
                 ;
                 ;------ string data area for message ------
                 ;
D016 0D0A476F    MESG:   DB    CR,LF,'Good Morning',CR,LF,EOS
D01A 6F64204D
D01E 6F726E69
D022 6E670D0A
D026 00
                 ;
D027                     END
```

```
LOAD OFFSET? 0 ↵ ………… オブジェクト・プログラムは，ORG疑似命令で指定したD000Hにそのままロードするので，
ロード終了               オフセットは0である
RETURN TO BASIC OR MONITOR (B/M) ?M ………… Mを入力してモニタへ

h]GD000 ……………… メッセージ出力プログラムを実行
Good Morning ……… 作成されたメッセージ出力プログラムによる表示

h]
```

# 6

# やさしい
# プログラミング実習

本章ではこれまでに解説した基礎知識を基に，やさしいプログラムを作ってみましょう．ここでは，目的どおり動作するプログラムを実現するためのアルゴリズム（筋道，構成，考え方などのこと）を考えることから，プログラミングの検討，ソース・プログラムの作成，アセンブル，ロード，実行までを実習解説します．

ここでの実行例は，作成されたプログラムをCP／M上で実行することを前提にしていますので，オブジェクト・プログラムのロード・アドレスを$0100_H$にしたり，CP／M内部のサブルーチンを利用したりしています．CP／Mをベースにしない場合は，ロード・アドレスや内部ルーチンの呼び出し方を，プログラムを実行する機種とその上の「基本ソフトウェア」に合わせて自由に変更してください．

本章では使用ツールが何であるかはあまり大きな意味をもちません．それよりも全体の流れをよく把握してください．たとえ使用するツールが学習用のツールであっても，ここでの実習にはほとんど支障はありません．しっかりと挑戦してみてください．

参考までに各種実行例として，マイクロソフト社のリロケータブル・マクロアセンブラ「M80」とリンクローダ「L80」を使った例と，CP／Mのシステムディスクに標準装備された8080アセンブラ「ASM」とローダ「LOAD」を使った例，それにBASIC上での例を示しておきます．

# 6/1 例題プログラムの仕様

本章で作成するプログラムは，たぶんみなさんも日常経験していると思いますが，画面によるメニュー選択のプログラムです．このプログラム名を「SELECT」とします．このメニュー選択のプログラムとは，いくつかの仕事のうちの1つを，ユーザーに選択させて実行するもので，これにはいろいろな形式のプログラムが考えられますが，ここでは最も簡単なものを作ってみましょう．次のような画面です．

```
 1 . Morning
 2 . Noon
 3 . Night
 4 . Bye

Input 1, 2, 3 or 4 >1

 * * Good Morning * *
```

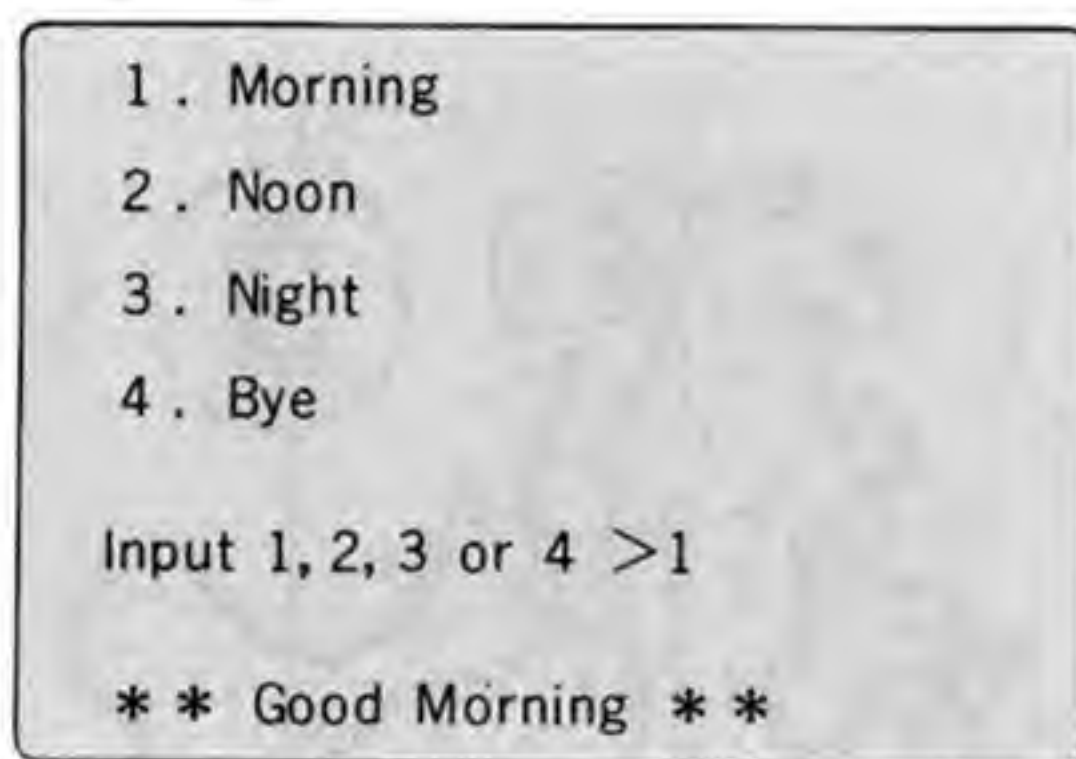



図6-1-1　例題プログラムのメニュー画面

プログラムを起動するとこのようなメニュー画面が表示されます．メニューには，1から4までの4項目があり，その番号をキー入力することにより，それぞれの番号に対する仕事が実行されます．仕事の内容にはいろいろなものが考えられますが，ここでは次のようなメッセージを表示するだけの単純なものにしておきます．

- **1を入力した場合**――Good morning
- **2を入力した場合**――Good afternoon
- **3を入力した場合**――Good night
- **4を入力した場合**――メッセージの表示はなく，当プログラムを終了する

それぞれの番号を入力することにより，該当するメッセージが表示されます．メッセージの表示が行われた後は，そのままポーズ状態になります．そこで，何らかのキー入力を行うことにより，プログラムは最初に戻り，再度メニューが表示されて，同じことを繰り返します．ただし4を入力するとこのプログラムは終了し，ここでの実習例では，CP/Mに戻ります．

プログラムの簡素化のために，入力した文字の取消しおよび再入力の機能は持たせていませんが，1～4以外の文字を入力した場合は[？]を表示させ，再入力となります．

# 6/2 アルゴリズムの構想

まずこのプログラムを実現するには，具体的にどのように考え，どのような方法をとるのがよいのか，つまりプログラムのアルゴリズムから考えなければなりません．その手法が決定したら，次はそれをどのようにプログラミングするかの検討です．そして，メインルーチンやサブルーチンなどを具体的にプログラミングしていきます．ではさっそく作業を開始してみましょう．

## メインルーチンについて

メインルーチンとは，プログラム全体の流れを決定する部分で，プログラム全体の骨子となるものです．ここでは，本章で例題とした「SELECT」プログラムを実現するための，メインルーチンから考えてみましょう．次に示すような流れのメインルーチンを考えてみました．

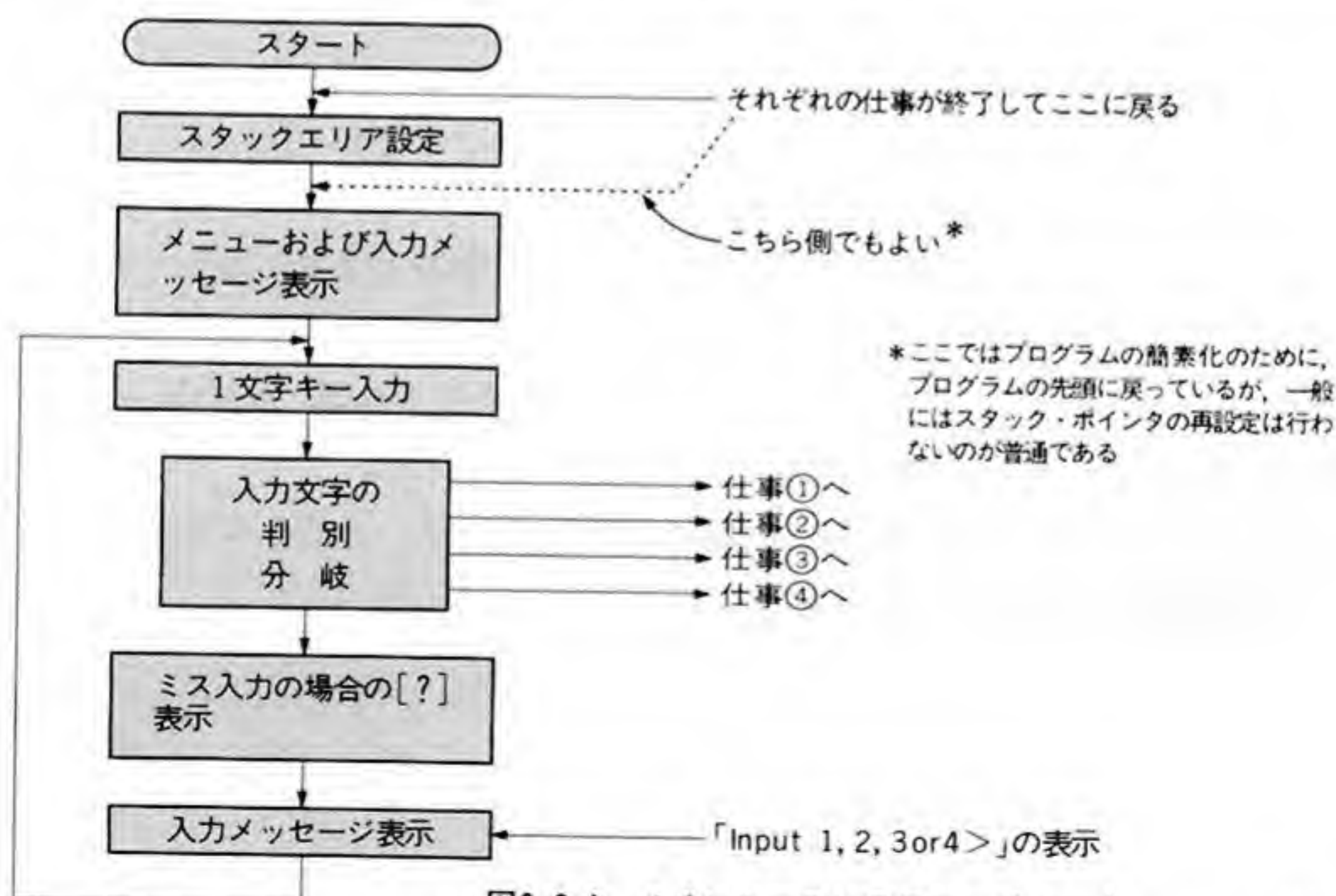



図6-2-1 メインルーチンのフローチャート

あまり難しいところはありませんが，このメインルーチン全体には次の要素が含まれています．

**(a) スタックエリアの設定**
**(b) メッセージの表示(メニューも含む)**
**(c) キーボードからの1文字入力**
**(d) 入力文字の判別および分岐**
**(e) 1文字出力([?]の表示)**

では，この一つひとつについて，簡単に検討してみましょう．

### (a) スタックエリアの設定

スタックエリアの設定に関しては本書では初登場ですが，これは通常，プログラムの開始時に設定するものです．この例題では特に設定しなくても問題なく動作しますが，スタックに関する知識は，アセンブラを利用する上でたいへん重要なので，本章の6.5で詳しく解説しています．

### (b) メッセージの表示(メニューも含む)

メッセージの表示には，今までの2章～5章で実習した文字列出力ルーチンがそのまま使えます．

### (c) キーボードからの1文字入力

キーボードからの文字入力については，CP/M内の1文字入力ルーチンを使いますが，そのルーチンの中身が具体的にどうなっているのかについてはここでは触れません．BASICをベースにする場合であれば，BASIC内の1文字入力ルーチンを使えばよいわけです．なおここでは，プログラムを簡素化するために入力した文字の訂正機能は付加しません．入力したものがすぐに次の処理に送られてしまいます．

### (d)　入力文字の判別および分岐

入力文字の判別と分岐の部分はこのプログラムの中核です．判別する項目が多い場合などは，いろいろなプログラミング上のテクニックがありますが，ここではごくあたりまえの手法を使います．

### (e)　1文字出力([？]の表示)

この処理も，(c)の場合と同様にCP／M内のルーチンを使います．BASICの場合はBASIC内のルーチンを使ってください．

以上のようなことから，メインルーチンは次の各ブロックで構成されることになります．

- **スタックエリアの設定**
- **メニューの表示**
- **1文字キー入力**
- **入力文字の判別・分岐**
- **ミス入力の場合の[？]表示**

当プログラムは，それぞれの仕事が実行されると再びプログラムの最初から繰り返され，初期のメニューが表示されます．これは，入力文字の判別に

より分岐が起こり，1～3の番号に対応した仕事のひとつが実行された後に，メインルーチンの冒頭に戻るJP命令によって行われます．この，プログラムの最初に戻るための処理は，分岐先の各仕事のルーチンで行っているためにメインルーチンには現れていません．

## 入力文字の判別および分岐

このメインルーチンの中でも特に重要な「入力文字判別および分岐」の部分のルーチンについて考えてみましょう．

これは，ユーザーが入力した，1～4の数字を判別し，それぞれの仕事のルーチンへ分岐させる部分です．

このプログラムの場合はわずか4つの項目の選択なので，特に難しく考える必要もなく，素直な流れを考えればよいでしょう．ここでは，次のようなフローを考えました．

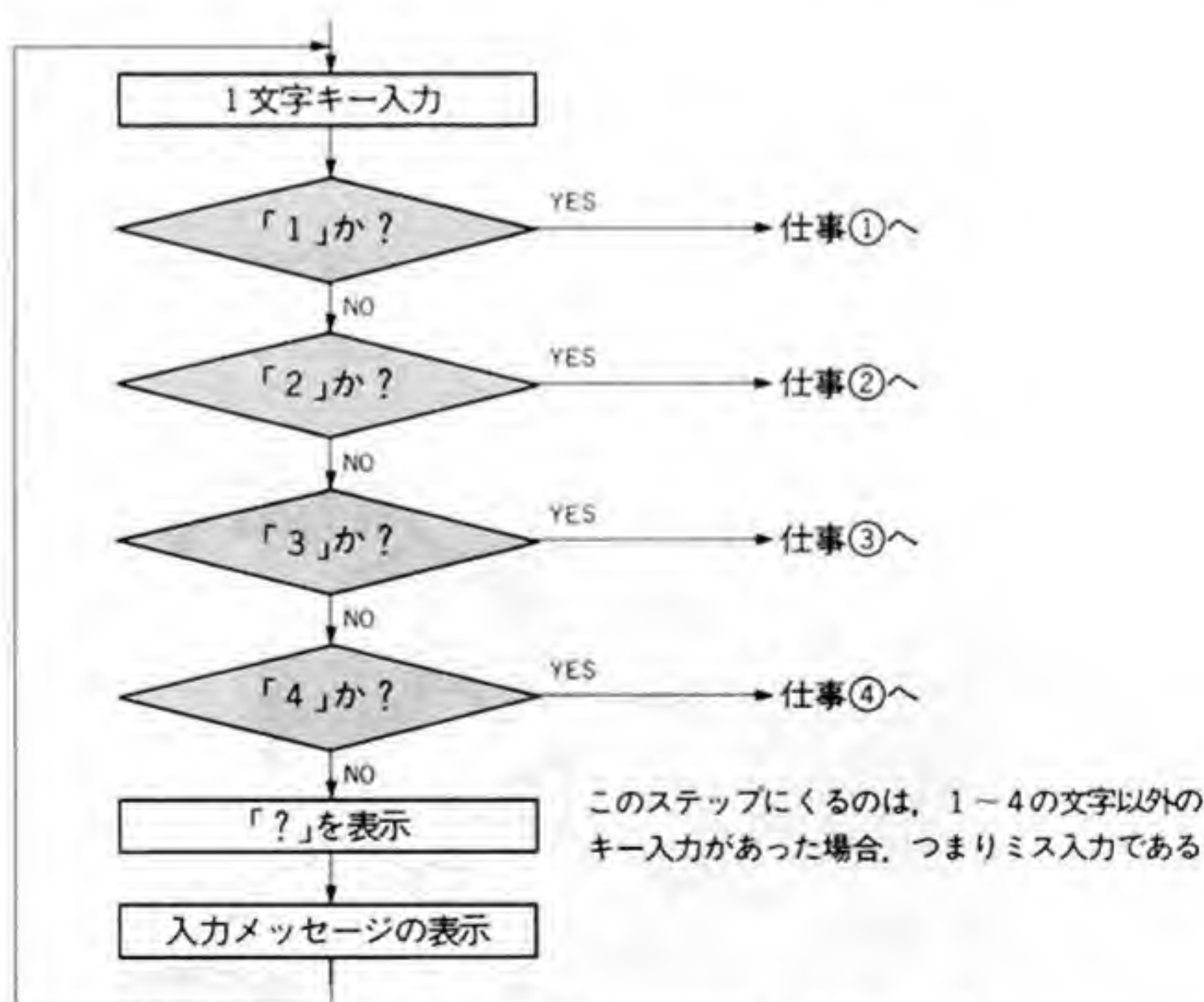



図6-2-2　入力文字の判別および分岐部のフローチャート

あまりエレガントではありませんが，選択肢が少ない場合はこのように，「1か？」，「2か？」，…というように，入力された文字の一致を順に確かめていく手法がよいでしょう．一致した場合は，その場からそれぞれの仕事のルーチンへジャンプします．もし一致しなかった（1～4以外が入力された）場合は，［？］を表示して再度キー入力待ちになります．

## 分岐先の仕事

分岐先の仕事には1～4の4種類があります．普通のプログラムでは，分岐先でいろいろと複雑な仕事を行わせるのでしょうが，このプログラムでは簡素化のために，1～3ではそれぞれ簡単なメッセージを表示するだけで，その後メインルーチンの頭に戻ります．4番目はこのプログラムを終了する仕事です．この部分のフローは次のようになるでしょう．

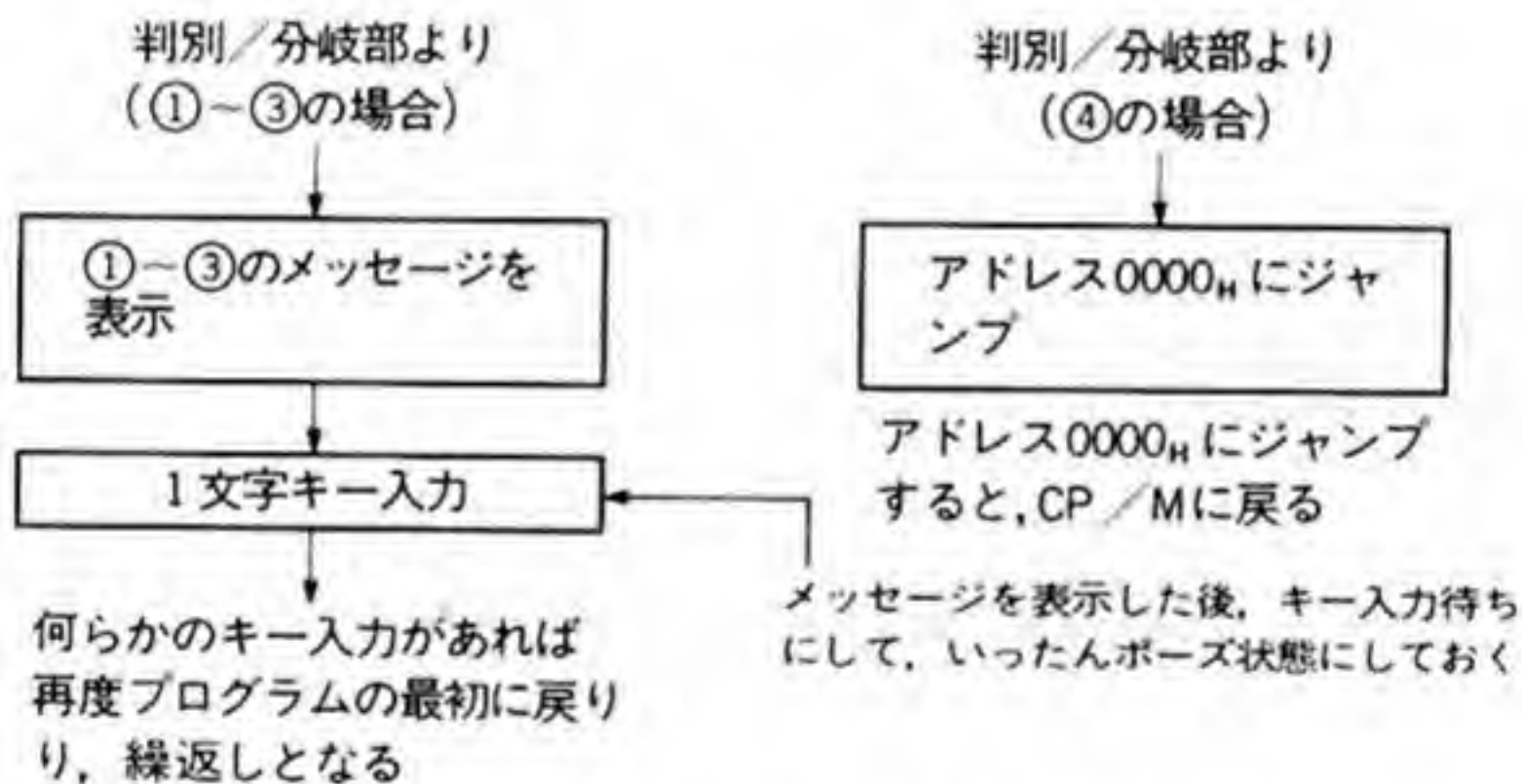



図6-2-3　分岐先の各仕事のフロー

## 全体の流れ

ではプログラムの全体のフローをまとめてみましょう．このプログラムのように小規模のものでは，プログラム全体のフローを1枚の紙の上に書き表すことができますが，規模が大きくなれば，それぞれの部分に分割したフローチャートを書くことになります．いちおう，次のような流れにすることにしましょう．

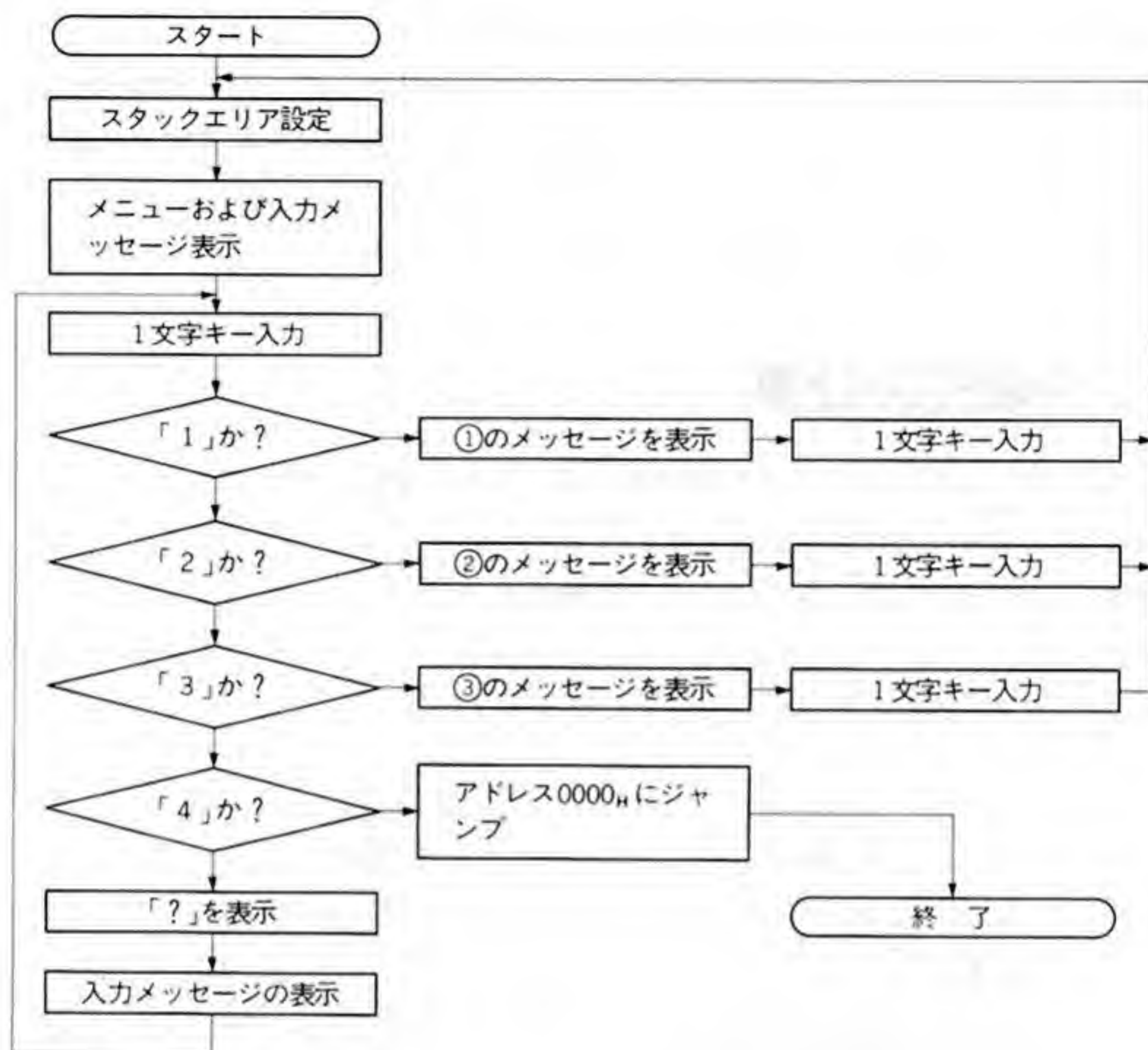



図6-2-4　当プログラム全体の流れ

# 6/3 プログラミング

本節では，先に示したフローチャートをアセンブラのソース・プログラムに置き換える作業，つまり「プログラミング」について解説します．

このプログラム全体は，プログラミング上では，メインルーチン，サブルーチン，および分岐先の仕事，の3つの部分に分けて考えることができます．サブルーチンに関しては，このプログラムでは，1文字キー入力ルーチン，1文字出力ルーチン，それに文字列出力ルーチンのサブルーチン化が可能なようです．

では，3つに分けた各部のプログラミングについて解説していきましょう．

## 各サブルーチン

まず当プログラムでサブルーチン化できるものを決定しておきましょう．次の3つが考えられます．

- **文字列出力サブルーチン**
- **1文字出力サブルーチン**
- **1文字入力サブルーチン**

では，これらのサブルーチンをプログラミングしてみましょう．

### ＊文字列出力サブルーチン／1文字出力サブルーチン

この2つのサブルーチンは，今までの例題で使っているおなじみのものをそのまま使います．文字列出力ルーチンは，今回はサブルーチン化して各所から共通に使えるようにしておきましょう．

1文字出力サブルーチンは文字列出力サブルーチンの内部で，コールされるサブルーチンとして使いますが，それとは別に，単独のサブルーチンとしても使われます．この1文字出力サブルーチンは，以前と同じくCP/M内部に用意されているユーザー・サービスルーチンを使いますが，BASICをベースにしている場合は，BASIC内の1文字出力ルーチンを利用してください．*

文字列出力サブルーチンに「MSGOUT」，1文字出力サブルーチンに「CHROUT」というラベルをつけると，これらは次のようにプログラムされます．

なおここでは，文字列出力サブルーチンを利用する場合は，表示しようとする文字列データが格納されているメモリの先頭番地をHLレジスタペアにセットしてからコールし，1文字出力サブルーチンを利用する場合は，表示する文字のアスキーコードをAレジスタにセットしてからコールすることを前提にしています．このために，1文字出力サブルーチンをコールする際に，HLレンジスタペアの値をPUSH，POPして保護しています．

```
MSGOUT:
        LD      A,(HL)
        OR      A
        RET     Z
        PUSH    HL
        CALL    CHROUT
        POP     HL
        INC     HL
        JP      MSGOUT
CHROUT:
        LD      C,2
        LD      E,A
        CALL    BDOS
        RET
```

*本章の6.4，およびAPPENDIX 1を参照．

ここでBDOSというのは，CP／Mの各ユーザー・サービスルーチン*をコールする場合の共通したエントリー・ポイント（呼び出しアドレス）で，$0005_H$番地にあたります.**

なお，この文字列出力サブルーチン「MSGOUT」は，CP／M内に用意されている１文字出力サブルーチンを利用して独自に作りましたが，CP／Mには文字列出力サブルーチンそのものが，ユーザー・サービスルーチンとして別に用意されています．本来はそれを利用する方が便利です．

### ＊１文字入力サブルーチン

キーボードから１文字入力するサブルーチンについても，CP／Mの内部に用意されているユーザー・サービスルーチンを利用します．このサブルーチンにつけるラベルを「CHRIN」としてそのプログラムを示します．

このサブルーチンを実行するとキー入力待ちとなり，キー入力が行われると入力文字が画面に表示され，そのアスキーコードがAレジスタにセットされてサブルーチンから戻ります．

```
CHRIN:
        LD      C,1
        CALL    BDOS
        RET
```

ここでは，プログラム全体をプログラミングする手順としてサブルーチンを先に決定しましたが，これは当プログラムが非常に小さく全体の構成が十分に見通せるためです．規模が大きい場合は，トップダウンといって，メインルーチンから構成していくことになるでしょう．

＊CP/Mのユーザー・サービスルーチン（システムコールと呼ぶ）については「APPENDIX 1」で解説している．
＊＊BDOSというラベルと$0005_H$という値は，次に解説するEQU定義で対応づけしている．

# EQU定義部およびメインルーチン

## * EQU 定義部

当プログラムで使用するシンボルの値を，メイン・プログラムの冒頭で定義しておきます．使用するシンボルは，次に示すように前章までの例題プログラムとまったく同じです．

```
BDOS    EQU    0005H
CR      EQU    0DH
LF      EQU    0AH
EOS     EQU    00
```

## *スタックエリア設定部

スタックについては本章の6．5で詳しく解説しますが，通常，スタックエリアは，全プログラムの最後部に置きます．スタックエリアは，プログラムの実行に際して，CALL 命令でサブルーチンに飛んだときにもとのルーチンに戻るためのアドレスを記録しておいたり，各レジスタの値を一時退避しておくためのメモリエリアです．サブルーチンのなかで，またサブルーチンがコールされたりすると(このようなことを「入れ子」とか「ネスティング」と呼ぶ)，その深さに応じてスタックエリアのバイト数が必要となります．

このプログラムではネスティングの深さは最高4レベル(3重のネスティングと PUSH 命令が1つ)なので，4×2バイト＝8バイトのエリアがあればよいのですが，いちおう8レベル，16バイトのスタックエリアを確保しておきましょう．

具体的には，擬似命令「DS」*により16バイトをプログラムの最後部に確保し，その最後部の次にスタック・ポインタを設定します．

この部分のプログラムと，そのときのメモリの様子を次に示します．なお，ここで使われている[＄]は,「ここのアドレス値」ということを表す記号です．詳しくは，8．3章を参照してください．

＊擬似命令については，8.4章を参照．

```
          LD      SP,STACK
                    ∫
          DS      16
STACK:    EQU     $

          END
```



( )内のアドレス値は，図6-4-2および,図6-4-5のアセンブルリストのプログラム(100H スタート）が，メモリ上にロードされたときのもの．図6-4-2のリストは，プログラムのスタートが（リスト上は）0000Hなので+100Hすることに注意

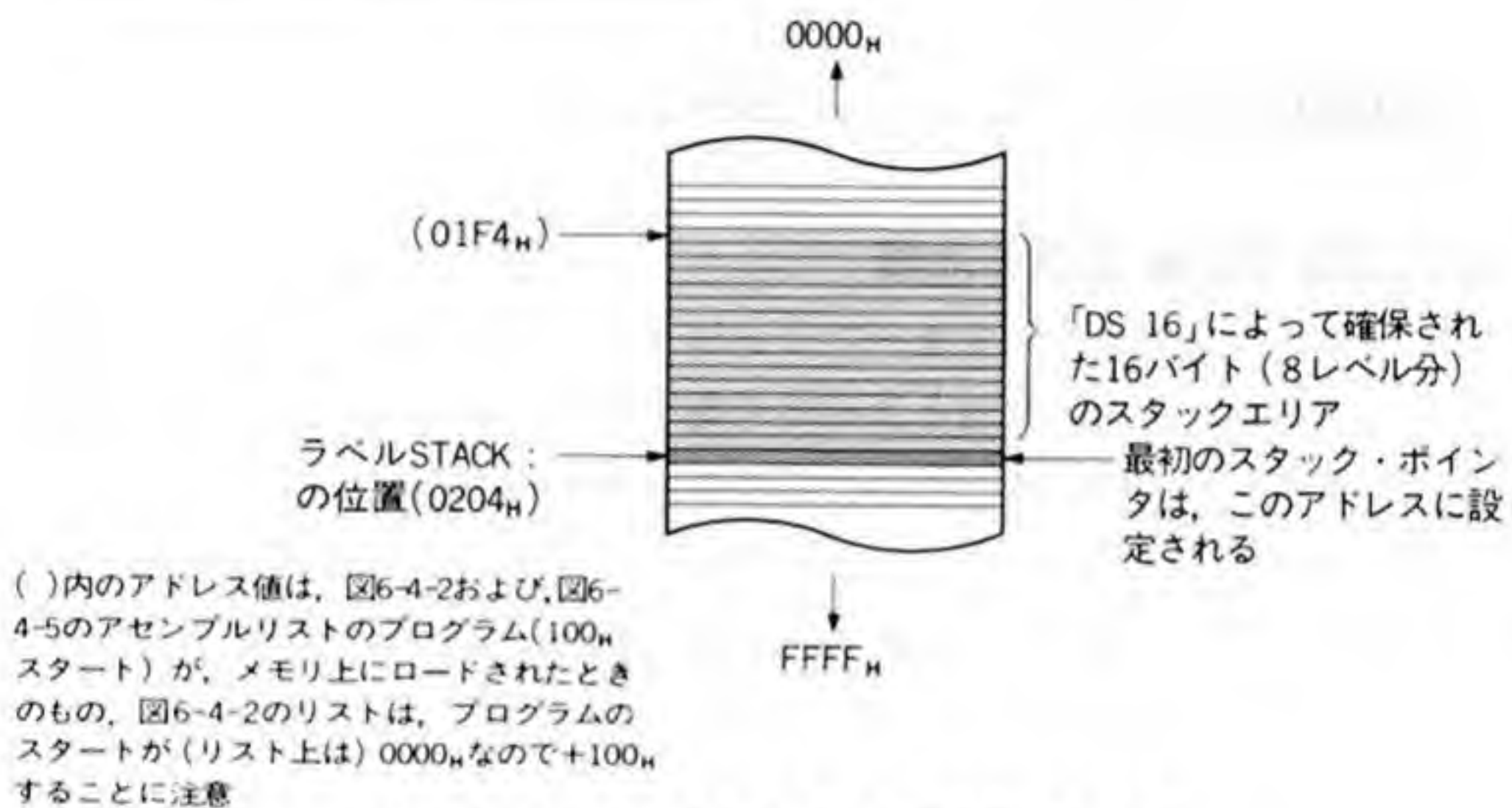



**図6-3-1　当プログラムのスタックエリアの様子**

## ＊メニュー表示部

ユーザーに項目の選択および入力を促すためのメニューを表示します．この文章のデータはソース・プログラムの後部に置き，「MSGMNU」というラベルをつけることにしましょう．

この文章データの部分は擬似命令「DB」を使います．表示する文章を[‘　’]で囲み，

```
MSGMNU: DB    '(メニューの文章データ)'
```

というステートメントをソース・プログラムの後部に用意しておきます．このデータを「文字列出力ルーチン」でCRTディスプレイに出力すればよいわけです．

では，メニュー表示部をプログラミングしてみましょう．HLレジスタペアにメニューの文章データの先頭アドレス(つまり，ラベル「MEGMNU」)をセットして，文字列出力サブルーチン「MSGOUT」をコールすればよいわけですから，プログラムは次のようになります．

```
            LD      HL,MSGMNU
            CALL    MSGOUT
              ∫
MSGMNU: DB   '(メニューの文章データ)'
```

**＊入力文字判別および分岐部**

キー入力された文字のアスキーコードはAレジスタにセットされていますので，このAレジスタの内容と1～4の数字のアスキーコードとを順に比較していきます．そして一致したところでそれぞれの仕事へ分岐させます．

Aレジスタと1バイトデータとの比較は「CP(コンペア)　xx」命令を使い，分岐には条件ジャンプ「JP　Z,xxxx」や「JP　NZ,xxxx」などを使えばよいでしょう．プログラムは次のようになります．

```
KEYIN:
            CALL    CHRIN
            CP      '1'
            JP      Z,JOB1
            CP      '2'
            JP      Z,JOB2
            CP      '3'
            JP      Z,JOB3
            CP      '4'
            JP      Z,JOB4
```

このプログラムでは，1～4のキー入力があった場合のみそれぞれJOB1～JOB4に分岐します．それ以外が入力された場合には，最後のステップを通

り抜けてしまいますので，このプログラムの最後にミス入力のための処理を加えておかなかればなりません．このため次の部分を付け加えます．

```
LD      A,'?'
CALL    CHROUT
LD      MSGINP *
CALL    MSGOUT
JP      KEYIN
```

この処理によって1～4以外のキー入力がされると[？]および入力メッセージが表示された後，再びラベル「KEYIN:」に戻り再度のキー入力が可能になります．

## 分岐先の仕事

1～3の入力であれば，JOB1～JOB3が実行されてそれぞれのメッセージが表示されます．4が入力されるとJOB4が実行され，当プログラムを終了してCP/Mに戻ります．

### * JOB1～JOB 3

これらは，入力された番号に対応したメッセージを表示するルーチンですので，メニュー表示の場合と同様の処理となります．表示する文章データは，それぞれDB擬似命令で，プログラムの後部に用意します．

表示の仕事が終わると1文字キー入力サブルーチンがコールされますので，そこで1度ポーズ状態になります．その後何らかのキー入力があれば，再びプログラムの頭に戻り，** メニューの表示が行われて，プログラムが繰り返されます．

＊先の説明では省略しているが，メニューの文章データの後半部分にMSGINPというラベルをつけている．ミス入力時には入力を促すために，[？]のほかにこの部分を表示する．詳しくはプログラムリストのデータ定義部分を参照．

＊＊ここでは，プログラムの先頭に戻り，スタック・ポインタの設定も含めてプログラム全体を再スタートしている．ただし普通のプログラムの場合は，スタック・ポインタの設定を何度も行わない．つまり，再スタートのためのジャンプ先はスタック・ポインタ設定部の直後になる．詳しくは図6-2-1のフローチャートを参照．

この部分のプログラムは次のようになります．

```
JOB1:
        LD      HL,MSGMNG
        CALL    MSGOUT
        CALL    CHRIN
        JP      START
JOB2:
        LD      HL,MSGNOO
        CALL    MSGOUT
        CALL    CHRIN
        JP      START
JOB3:
        LD      HL,MSGNIT
        CALL    MSGOUT
        CALL    CHRIN
        JP      START
          ∫
MSGMNG: DB  CR,LF,'** Good morning**',CR,LF,EOS
MSGNOO: DB  CR,LF,'** Good afternoon**',CR,LF,EOS
MSGNIT: DB  CR,LF,'** Good night  **',CR,LF,EOS
```

### ＊JOB 4

当プログラムから，実行のベースになっているCP/MやBASICに戻るためのルーチンです．戻る方法は，スタック・ポインタの扱い方により異なりますが，CP/MではRET命令を実行したり，あるいはアドレス$0000_H$にジャンプすることにより可能です．BASIC上で実行した場合でも，RET命令やRST命令，あるいは「コールド・スタート」と呼ばれるルーチンにジャンプすることにより，戻ることができます．$N_{88}$-BASICの場合は，「RST　38 H」

を使っていますが，これはそれぞれのBASICにより異なりますので注意してください．

CP/M上では，実行したプログラムの中で新たにスタック・ポインタの設定をしていないのであれば，RET命令を実行するだけで戻ることができます．その場合，CP/Mからみると，プログラムは，サブルーチンコールによって実行されているということになります．

しかしここでのプログラムは，スタック・ポインタを操作してスタックエリアを設定していますので，RET命令だけでは戻ることはできません．

このような場合，CP/Mには「リブート」*と呼ばれるCP/Mへの復帰手段が用意されています．これはアドレス$0000_H$にジャンプするだけの簡単なもので，スタック・ポインタには関係なく無条件でユーザー・プログラムからCP/Mに戻ることができます．

いずれにしても，これらはスタック・ポインタに深く関係しますので，スタックについてよく理解しておいてください．

JOB4のルーチンはたいへん簡単ですが，次のようになります．

```
JOB4 :      JP      0000
```

＊コールド・スタートのこと．

## 全ソース・プログラム

今までに解説した各ルーチンから，アセンブル可能なソース・プログラムをエディタを使って作成します．できあがったソース・プログラムを次に示します．

図6-3-2 全ソース・プログラム

```
;-------------------------------------;
;        MENU SELECT PROGRAM          ;
;-------------------------------------;
;
BDOS    EQU     0005H
CR      EQU     0DH
LF      EQU     0AH
EOS     EQU     00
;
        ORG     100H
;
;----------- main routine ------------
START:
        LD      SP,STACK      ……スタックエリア設定
        LD      HL,MSGMNU     | メニューおよび入力メッセージ表示
        CALL    MSGOUT        |
KEYIN:
        CALL    CHRIN         ……1文字キー入力
        CP      '1'           |
        JP      Z,JOB1        |
        CP      '2'           |
        JP      Z,JOB2        | 入力文字の判別/分岐
        CP      '3'           |
        JP      Z,JOB3        |
        CP      '4'           |
        JP      Z,JOB4        |
;
        LD      A,'?'         | このステップにくるのは1～4以外の文字をミス入力した
        CALL    CHROUT        | 場合なので「?」を表示
        LD      HL,MSGINP     | 入力メッセージの表示
        CALL    MSGOUT        |
        JP      KEYIN         ……再度1文字キー入力へ
;
JOB1:
        LD      HL,MSGMNG     | メッセージの表示           |
        CALL    MSGOUT        |                            | 仕事①
        CALL    CHRIN         ……いったんポーズ状態にするための1文字キー入力 |
        JP      START         ……プログラムの最初から再スタート |
;
JOB2:
        LD      HL,MSGNOO     |
        CALL    MSGOUT        |
        CALL    CHRIN         | 仕事②
        JP      START         |
```

```
;
JOB3:
        LD      HL,MSGNIT      |
        CALL    MSGOUT         |
        CALL    CHRIN          |  仕事③
        JP      START          |
;
JOB4:
        JP      0000 ……………当プログラムを終了してCP/Mに戻る：仕事④
;
;
;------------ subroutine -------------
;       message string out subroutine
MSGOUT:
        LD      A,(HL)
        OR      A
        RET     Z
        PUSH    HL
        CALL    CHROUT
        POP     HL
        INC     HL
        JP      MSGOUT                         文字列出力サブルーチン
;
;------ 1 character out subroutine
CHROUT:
        LD      C,2     |
        LD      E,A     |  CP/Mの内部ルーチンを利用
        CALL    BDOS    |  した1文字出力サブルーチン
        RET             |
;
;------ 1 character key input subroutine
CHRIN:
        LD      C,1     |
        CALL    BDOS    |  CP/Mの内部ルーチンを利用した1文字入力サブルーチン
        RET             |
;
;
;------ string data and stack area ------
MSGMNU: DB      CR,LF,'1. Morning'
        DB      CR,LF,'2. Noon'
        DB      CR,LF,'3. Night'
        DB      CR,LF,'4. Bye'
MSGINP: DB      CR,LF,LF,'Input 1,2,3 or 4  >',EOS
;
MSGMNG: DB      CR,LF,LF,'** Good morning **',CR,LF,EOS
MSGNOO: DB      CR,LF,LF,'** Good afternoon **',CR,LF,EOS
MSGNIT: DB      CR,LF,LF,'** Good night **',CR,LF,EOS
;
        DS      16  |  この16バイトが8レベルのスタックエリア
STACK   EQU     $   |
;
        END
```

各メッセージデータ，それぞれ指定されたラベルから，「EOS（コード00）までが文字列出力サブルーチンで表示される

# 6/4 アセンブル，ロードおよび実行

では，できあがったソース・プログラム，ファイル名「SELECT.MAC」をアセンブルし，生成されたオブジェクト・プログラムに対してローダを実行した後，プログラムを走らせてみましょう．ここでは，マイクロソフト社のリロケータブル・マクロアセンブラ「M80」を使った実行例を示します．*

## ＊アセンブラの実行

アセンブラ「M80」を実行します．ソース・プログラム「SELECT.MAC」がアセンブラに入力され，そこで処理された後，オブジェクト・プログラムとアセンブルリストのファイルが生成される様子を次の実行例で示します．

図6-4-1　アセンブラの実行

```
A>DIR SELECT.* ↵    ……ファイル「SELECT」に関するすべてのファイル名をタイプアウトする
A: SELECT   MAC
   ソース・プログラムのみ存在している
A>M80 SELECT,SELECT=SELECT/Z ↵    ……アセンブラ「M80」の実行．最後の「/Z」は，Z-80ニーモニック
                                      をアセンブルするためのスイッチ
No Fatal error(s)
   アセンブル・エラーなくアセンブル終了
A>DIR SELECT.* ↵
A: SELECT   MAC : SELECT   PRN : SELECT   REL
    ソース・プログラム   生成されたアセンブル   生成されたリロケータブル・
A>                      リスト               オブジェクト・プログラム
```

エラーはなくアセンブルは成功し，リロケータブルなオブジェクト・プログラムと，アセンブルリストのファイルがディスク上に生成されています．このアセンブルリストをタイプアウトして次に示します．

＊M80の場合，そのソース・プログラムのファイル名は，必ず「.MAC」でなければならない．M80でアセンブルする場合は，ソース・プログラムの「ORG 100H」の部分を「CSEG」に変更すること（アセンブルリスト参照）．これについては11章「リロケータブル・マクロアセンブラの概念と使い方」で解説する．

図6-4-2　アセンブルにより生成されたアセンブルリスト

```
        MACRO-80 3.44   09-Dec-81       PAGE    1

                                ;-------------------------------------------;
                                ;           MENU SELECT PROGRAM             ;
                                ;-------------------------------------------;
                                ;
0005                            BDOS    EQU     0005H
000D                            CR      EQU     0DH
000A                            LF      EQU     0AH
0000                            EOS     EQU     00
                                ;
0000'                                   CSEG
                                ;
                                ;------------ main routine -------------
0000'                           START:
0000'     31 0104'                      LD      SP,STACK
0003'     21 006E'                      LD      HL,MSGMNU
0006'     CD 0055'                      CALL    MSGOUT
0009'                           KEYIN:
0009'     CD 0068'                      CALL    CHRIN
000C'     FE 31                         CP      '1'
000E'     CA 002E'                      JP      Z,JOB1
0011'     FE 32                         CP      '2'
0013'     CA 003A'                      JP      Z,JOB2
0016'     FE 33                         CP      '3'
0018'     CA 0046'                      JP      Z,JOB3
001B'     FE 34                         CP      '4'
001D'     CA 0052'                      JP      Z,JOB4
                                ;
0020'     3E 3F                         LD      A,'?'
0022'     CD 0061'                      CALL    CHROUT
0025'     21 0095'                      LD      HL,MSGINP
0028'     CD 0055'                      CALL    MSGOUT
002B'     C3 0009'                      JP      KEYIN
                                ;
002E'                           JOB1:
002E'     21 00AC'                      LD      HL,MSGMNG
0031'     CD 0055'                      CALL    MSGOUT
0034'     CD 0068'                      CALL    CHRIN
0037'     C3 0000'                      JP      START
                                ;
003A'                           JOB2:
003A'     21 00C4'                      LD      HL,MSGNOO
003D'     CD 0055'                      CALL    MSGOUT
0040'     CD 0068'                      CALL    CHRIN
0043'     C3 0000'                      JP      START
                                ;
0046'                           JOB3:
0046'     21 00DE'                      LD      HL,MSGNIT
0049'     CD 0055'                      CALL    MSGOUT
004C'     CD 0068'                      CALL    CHRIN
004F'     C3 0000'*                     JP      START
                                ;
0052'*                          JOB4:
0052'     C3 0000                       JP      0000
                                ;
                                ;
```

M-80によるアセンブルリストでは，オブジェクト部のアドレスや2バイトの数値を表す部分(スペースによる区切りがなく，2バイトが連続している部分)が，メモリ上にロードされる順序ではなく，「読む順序」になっていることに注意．例えばこの部分など（0104'）

CSEG …「M80」でアセンブルする場合は，「ORG 100H」をこのように書き換えておくこと

* アドレス値につけられた[ ' ]記号は，確定アドレスではなく，相対的な値であることを示す．つまりリロケータブルであるということ

```
                              ;------------ subroutine -------------
                              :       message string out sub routine
0055'                         MSGOUT:
0055'    7E                           LD      A,(HL)
0056'    B7                           OR      A
0057'    C8                           RET     Z
0058'    E5                           PUSH    HL
0059'    CD 0061'                     CALL    CHROUT
005C'    E1                           POP     HL
005D'    23                           INC     HL
005E'    C3 0055'                     JP      MSGOUT
                              ;
                              ;------ 1 character out subroutine
0061'                         CHROUT:
0061'    0E 02                        LD      C,2
0063'    5F                           LD      E,A
0064'    CD 0005                      CALL    BDOS
0067'    C9                           RET
                              ;
                              ;------ 1 character key input subroutine
0068'                         CHRIN:
0068'    0E 01                        LD      C,1
006A'    CD 0005                      CALL    BDOS
006D'    C9                           RET
                              ;
                              ;
                              ;------ string data and stack area ------
006E'    0D 0A 31 2E          MSGMNU: DB      CR,LF,'1. Morning'
0072'    20 4D 6F 72
0076'    6E 69 6E 67
007A'    0D 0A 32 2E                  DB      CR,LF,'2. Noon'
007E'    20 4E 6F 6F
0082'    6E
0083'    0D 0A 33 2E                  DB      CR,LF,'3. Night'
0087'    20 4E 69 67
008B'    68 74
008D'    0D 0A 34 2E                  DB      CR,LF,'4. Bye'
0091'    20 42 79 65
0095'    0D 0A 0A 49          MSGINP: DB      CR,LF,LF,'Input 1,2,3 or 4  >',
0099'    6E 70 75 74                                                  EOS
009D'    20 31 2C 32
00A1'    2C 33 20 6F
00A5'    72 20 34 20
00A9'    20 3E 00
                              ;
00AC'    0D 0A 0A 2A          MSGMNG: DB      CR,LF,LF,'** Good morning **',
00B0'    2A 20 47 6F                                          CR,LF,EOS
00B4'    6F 64 20 6D
00B8'    6F 72 6E 69
00BC'    6E 67 20 2A
00C0'    2A 0D 0A 00
00C4'    0D 0A 0A 2A          MSGNOO: DB      CR,LF,LF,'** Good afternoon **'
00C8'    2A 20 47 6F                                          ,CR,LF,EOS
00CC'    6F 64 20 61
00D0'    66 74 65 72
00D4'    6E 6F 6F 6E
00D8'    20 2A 2A 0D
00DC'    0A 00
```

```
00DE'    0D 0A 0A 2A    MSGNIT: DB      CR,LF,LF,'** Good night **',CR,
                                                  LF,EOS
00E2'    2A 20 47 6F
00E6'    6F 64 20 6E
00EA'    69 67 68 74
00EE'    20 2A 2A 0D
00F2'    0A 00
                        ;
00F4'                           DS      16
0104'                   STACK   EQU     $
                        ;
                                END
```

ロード・アドレス　オブジェクト・プログラム　ソース・プログラム

```
        MACRO-80 3.44   09-Dec-81       PAGE    S


Macros:

Symbols:
0005    BDOS            0068'   CHRIN           0061'   CHROUT
000D    CR              0000    EOS             002E'   JOB1
003A'   JOB2            0046'   JOB3            0052'   JOB4
0009'   KEYIN           000A    LF              0095'   MSGINP
00AC'   MSGMNG          006E'   MSGMNU          00DE'   MSGNIT
00C4'   MSGNOO          0055'   MSGOUT          0104'   STACK
0000'   START

No Fatal error(s)
```

シンボルの一覧表

## *ローダの実行

生成されたオブジェクト・プログラムはリロケータブル形式(.REL)なので，メモリ上にロードして実行する際の絶対的なアドレス値が固定されているわけではありません．従って，これをリンクローダ「L80」によって任意のロード・アドレスをもつ，実行可能な純マシンコードのオブジェクト・プログラムに変換することが可能です．その実行例は2通りありますが，それを次に示します．

最初の例は実行可能なオブジェクト・プログラムをリンクローダから直接生成する場合の例であり，次は，リンクローダからインテルHEX形式のオブジェクト・プログラムを生成した後にCP/Mのローダ「LOAD」で実行可能なオブジェクト・プログラムに変換する場合の例です．

図6-4-3(a) リンクローダの実行
(実行可能なオブジェクト・プログラムを直接生成する例)

```
A>DIR SELECT.*        ……リンクローダの実行前にすべての「SELECT」ファイルを確認
A: SELECT   MAC : SELECT   PRN : SELECT   REL
                              L80に入力されるリロケータブル・オブジェクト・ファイル
A>L80 /P:100,SELECT,SELECT/N/E   ……リンクローダの実行．スタート・アドレス0100Hの実行可能なオブジェクト・プログラムを生成する
Link-80  3.44  09-Dec-81  Copyright (c) 1981 Microsoft
                エンド・アドレス
Data    0100   0204     <   260>
 プログラムのスタート・アドレス      プログラム全体のバイト数(10進数)
40758 Bytes Free
[0000   0204      2]
 リンクローダの実行終了
A>DIR SELECT.*        ……リンクローダ実行後にすべての「SELECT」ファイルを確認
A: SELECT   MAC : SELECT   PRN : SELECT   REL : SELECT   COM
A>                                              生成された実行可能な
                                                純マシン語のファイル
```

図6-4-3(b) リンクローダの実行
(HEX形式のオブジェクト・プログラムを生成してから実行可能なオブジェクト・プログラムに変換する例)

```
A>DIR SELECT.*
```
……リンクローダ実行前にすべての「SELECT」ファイルを確認

```
A: SELECT   MAC : SELECT   PRN : SELECT   REL
```
このリロケータブル・オブジェクト・ファイルがL80に入力される

```
A>L80 /P:100,SELECT,SELECT/N/X/E
```
……L80を実行して,0100Hスタートのインテル HEX 形式のオブジェクトを生成する

```
Link-80  3.44  09-Dec-81  Copyright (c) 1981 Microsoft

Data    0100    0204    <  260>
```
0100: プログラムのスタート・アドレス　0204: エンド・アドレス　260: プログラム全体のバイト数(10進数)

```
40758 Bytes Free
[0000   0204        2]


A>DIR SELECT.*
```
……リンクローダ実行後にすべての「SELECT」ファイルを確認

```
A: SELECT   MAC : SELECT   PRN : SELECT   REL : SELECT   HEX
```
L80により生成されたインテルHEX形式のオブジェクト・プログラム

```
A>TYPE SELECT.HEX
```
……生成されたインテルHEX形式のオブジェクトファイルをタイプアウトする

```
:2001000031040221 6E01CD5501CD6801FE31CA2E01FE32CA3A01FE33CA4601FE34CA5201D1
:200120003E3FCD610121950 1CD5501C3090121AC01CD5501CD6801C3000121C401CD550178
:20014000CD6801C3000121DE01CD5501CD6801C30001C300007EB7C8E5CD6101E123C35598
:20016000010E025FCD0500C90E01CD0500C90D0A312E204D6F726E696E670D0A322E204E75
:200180006F6F6E0D0A332E204E696768740D0A342E204279650D0A0A496E70757420312C1A
:2001A000322C33206F72203420203E000D0A0A2A2A20476F6F64206D6F726E696E67202A59
:2001C0002A0D0A000D0A0A2A2A20476F6F642061667465726E6F6F6E202A2A0D0A000D0A32
:2001E0000A2A2A20476F6F64206E69676874202A2A0D0A00043AEC3CB7C87BB7CAE12BFE48
:040200000ADC9D0473
:00000001FF
```
ロード・アドレス　チェックサム　オブジェクトデータ　チェックサム

```
A>LOAD SELECT
```
……CP/Mのローダ「LOAD」の実行, HEX形式のオブジェクトから, 実行可能な純マシン語のプログラムファイルを生成する

```
FIRST ADDRESS 0100
LAST  ADDRESS 0203
BYTES READ    0104
RECORDS WRITTEN 03
```
ローダの実行終了

```
A>DIR SELECT.*
```
……すべての「SELECT」ファイルを確認する

```
A: SELECT   MAC : SELECT   PRN : SELECT   REL : SELECT   HEX
A: SELECT   COM
A>
```
生成された実行可能な純マシン語のプログラム

### ＊完成プログラムの実行

完成した実行可能なプログラム「SELECT.COM」を実行してみましょう．実行するのは簡単で「.COM」を省いたファイル名「SELECT」をキー入力してリターンするだけです．その実行例を示します．

図6-4-4　完成したプログラムの実行

## CP/MのASM, LOADでの実行例

CP/Mのアセンブラ「ASM」は8080 CPUのアセンブラです．今までのソース・プログラムはZ-80用のザイログ表記のニーモニックで書いてありますので，このままでは「ASM」でアセンブルできません．

これを使うには8080用のニーモニックで書き直さなければなりませんが，例題のソース・プログラムで使っているZ-80の命令はすべて8080とコンパチブルのものだけですので，そっくりそのまま1対1で8080用の命令に置き換えることができます．ここでは，8080用に書き直したソース・プログラム

(アセンブルリスト)を次に示します.

図6-4-5　8080CPU用に書き直したプログラム

```
FILE: SELECT80 PRN                          PAGE 001
Z-80用のSELECTプログラムを，そのまま8080用に書き直してアセンブルしたアセンブルリスト
                 ;--------------------------------------;
                 ;          MENU SELECT PROGRAM          ;
                 ;--------------------------------------;
                 ;
 0005 =          BDOS    EQU     0005H
 000D =          CR      EQU     0DH
 000A =          LF      EQU     0AH
 0000 =          EOS     EQU     00
                 ;
 0100                    ORG     100H
                 ;
                 ;------------ main routine ------------
                 START:
 0100 310402             LXI     SP,STACK
 0103 216E01             LXI     H,MSGMNU
 0106 CD5501             CALL    MSGOUT
                 KEYIN:
 0109 CD6801             CALL    CHRIN
 010C FE31               CPI     '1'
 010E CA2E01             JZ      JOB1
 0111 FE32               CPI     '2'
 0113 CA3A01             JZ      JOB2
 0116 FE33               CPI     '3'
 0118 CA4601             JZ      JOB3
 011B FE34               CPI     '4'
 011D CA5201             JZ      JOB4
                 ;
 0120 3E3F               MVI     A,'?'
 0122 CD6101             CALL    CHROUT
 0125 219501             LXI     H,MSGINP
 0128 CD5501             CALL    MSGOUT
 012B C30901             JMP     KEYIN
                 ;
                 JOB1:
 012E 21AC01             LXI     H,MSGMNG
 0131 CD5501             CALL    MSGOUT
 0134 CD6801             CALL    CHRIN
 0137 C30001             JMP     START
                 ;
                 JOB2:
 013A 21C401             LXI     H,MSGNOO
 013D CD5501             CALL    MSGOUT
 0140 CD6801             CALL    CHRIN
 0143 C30001             JMP     START
                 ;
                 JOB3:
 0146 21DE01             LXI     H,MSGNIT
 0149 CD5501             CALL    MSGOUT
 014C CD6801             CALL    CHRIN
 014F C30001             JMP     START
                 ;
                 JOB4:
 0152 C30000             JMP     0000
                 ;
                 ;
```

```
                  ;------------ subroutine -------------
                  ;       message string out subroutine
                  MSGOUT:
0155 7E                   MOV     A,M
0156 B7                   ORA     A
0157 C8                   RZ
0158 E5                   PUSH    H
0159 CD6101               CALL    CHROUT
015C E1                   POP     H
015D 23                   INX     H
015E C35501               JMP     MSGOUT
                  ;
                  ;------ 1 character out subroutine
                  CHROUT:
0161 0E02                 MVI     C,2
0163 5F                   MOV     E,A
0164 CD0500               CALL    BDOS
0167 C9                   RET
                  ;
                  ;------ 1 character key input subroutine
                  CHRIN:
0168 0E01                 MVI     C,1
016A CD0500               CALL    BDOS
016D C9                   RET
                  ;
                  ;
                  ;------ string data and stack area ------
016E 0D0A312E20MSGMNU: DB       CR,LF,'1. Morning'
017A 0D0A322E20        DB       CR,LF,'2. Noon'
0183 0D0A332E20        DB       CR,LF,'3. Night'
018D 0D0A342E20        DB       CR,LF,'4. Bye'
0195 0D0A0A496EMSGINP: DB       CR,LF,LF,'Input 1,2,3 or 4  >',EOS
                  ;
01AC 0D0A0A2A2AMSGMNG: DB       CR,LF,LF,'** Good morning **',CR,LF,EOS
01C4 0D0A0A2A2AMSGNOO: DB       CR,LF,LF,'** Good afternoon **',CR,LF,EOS
01DE 0D0A0A2A2AMSGNIT: DB       CR,LF,LF,'** Good night **',CR,LF,EOS
                  ;
01F4                      DS      16
0204 =            STACK   EQU     $
                  ;
0204                      END
```

CP/Mのアセンブラ「ASM」によるアセンブルリストでは，DBなどによるオブジェクトコードの表示は，先頭の5バイトのみが表示され，それ以上は省略されている

ではこの8080用に書き直されたソース・プログラム「SELECT80.ASM」*を，CP/Mのアセンブラ「ASM」とローダ「LOAD」を使って処理してから完成したプログラムを実行するまでの実行例を示しましょう．

*アセンブラ「ASM」のソース・プログラムのファイル名は，必ず「.ASM」でなければならない．

図6-4-6　8080アセンブラ「ASM」によるアセンブル処理

```
A>DIR SELECT80.* ↵ ……………… 実行前のSELECT80ファイルの確認

A: SELECT80 ASM
   ソースファイルのみ存在

A>ASM SELECT80 ↵ ……………… CP/Mのアセンブラ「ASM」の実行

CP/M ASSEMBLER - VER 2.0
0204
001H USE FACTOR
END OF ASSEMBLY
  アセンブル・エラーなくアセンブル終了

A>DIR SELECT80.* ↵ ……………… SELECT80ファイルの確認

A: SELECT80 ASM : SELECT80 PRN : SELECT80 HEX
                  生成されたアセンブル・  生成されたインテルHEX形式の
                  リスト・ファイル        オブジェクトファイル
A>LOAD SELECT80 ↵ ……………… ローダ「LOAD」の実行

FIRST ADDRESS 0100
LAST  ADDRESS 01F3
BYTES READ    00F4
RECORDS WRITTEN 02
  ロード終了

A>DIR SELECT80.* ↵ ……………… SELECT80ファイルの確認

A: SELECT80 ASM : SELECT80 PRN : SELECT80 HEX : SELECT80 COM
                                                生成された実行可能な
A>                                              オブジェクトファイル
```

```
A>SELECT80 ↵ ……………… 完成したプログラムSELECT80の実行

1. Morning
2. Noon          Z-80によるものとまったく同様に実行される
3. Night
4. Bye

Input 1,2,3 or 4  >1

** Good morning **
x
1. Morning
2. Noon
3. Ni...
   Good night **
x
1. Morning
2. Noon
3. Night
4. Bye

Input 1,2,3 or 4  >4

A>
```

# BASICをベースとする場合の実行例

これまでは，完成したプログラムをCP/M上で実行するための開発例を示してきました．今度はBASIC上での実行を目的とした同じプログラムを作成してみましょう．

このプログラムは，PC-8801（mkIIを含む）の$N_{88}$-BASICを対象としたもので，その開発にはCP/M上の「M80」と「L80」を使います．これらのツールで，アドレス$D000_H$スタートのインテルHEX形式のオブジェクト・プログラムを作成した後，5.2章で紹介したDUAD-88Dを使ってメモリへロードし実行してみましょう．

このような手順にしたのは，インテルHEX形式のオブジェクト・プログラムが，CP/MでもDUADでも共通に使えることを確認して，これがオブジェクト・プログラムの代表的な形式であることを認識するという意味もあります．もちろん最初から「DUAD-88D」だけで開発してもかまいません．

まず，ソース・プログラムにいくつかの変更点*がありますので，それを次のアセンブルリストで示します．

*5.2章にもDUADによるソース・プログラムの例がある．

**図6-4-7 $N_{88}$-BASIC用ソース・プログラム**

```
        MACRO-80 3.44   09-Dec-81       PAGE    1


                                ;-----------------------------------------;
                                ;          MENU SELECT PROGRAM            ;
                                ;-----------------------------------------;
                                ;                      1文字キー入力のROM内ルーチ
3583                            PCIN    EQU     3583H  …ンのエントリ・ポイント
3E0D                            PCOUT   EQU     3E0DH  …1文字出力のROM内ルーチンの
000D                            CR      EQU     0DH      エントリ・ポイント
000A                            LF      EQU     0AH
0000                            EOS     EQU     00
                                ;
0000'                                   CSEG
                                ;
                                ;------------ main routine -------------
0000'                           START:          (スタック・ポインタの設定はしていない)
0000'   21 0067'                        LD      HL,MSGMNU
0003'   CD 0050'                        CALL    MSGOUT
0006'                           KEYIN:
0006'   CD 0060'                        CALL    CHRIN
0009'   FE 31                           CP      '1'
000B'   CA 002B'                        JP      Z,JOB1
000E'   FE 32                           CP      '2'
0010'   CA 0037'                        JP      Z,JOB2
0013'   FE 33                           CP      '3'
0015'   CA 0043'                        JP      Z,JOB3
0018'   FE 34                           CP      '4'
001A'   CA 004F'                        JP      Z,JOB4
                                ;
001D'   3E 3F                           LD      A,'?'
001F'   CD 005C'                        CALL    CHROUT
0022'   21 008E'                        LD      HL,MSGINP
0025'   CD 0050'                        CALL    MSGOUT
0028'   C3 0006'                        JP      KEYIN
                                ;
002B'                           JOB1:
002B'   21 00A5'                        LD      HL,MSGMNG
002E'   CD 0050'                        CALL    MSGOUT
0031'   CD 0060'                        CALL    CHRIN
0034'   C3 0000'                        JP      START
                                ;
0037'                           JOB2:
0037'   21 00BD'                        LD      HL,MSGNOO
003A'   CD 0050'                        CALL    MSGOUT
003D'   CD 0060'                        CALL    CHRIN
0040'   C3 0000'                        JP      START
                                ;
0043'                           JOB3:
0043'   21 00D7'                        LD      HL,MSGNIT
0046'   CD 0050'                        CALL    MSGOUT
0049'   CD 0060'                        CALL    CHRIN
004C'   C3 0000'                        JP      START
                                ;
004F'                           JOB4:
004F'   FF                              RST     38H    …モニタに戻るためのリスタート命令
                                ;
                                ;
```

```
                        ;------------ subroutine -------------
                        ;       message string out subroutine
0050'                   MSGOUT:
0050'   7E                      LD      A,(HL)
0051'   B7                      OR      A
0052'   C8                      RET     Z
0053'   E5                      PUSH    HL
0054'   CD 005C'                CALL    CHROUT
0057'   E1                      POP     HL
0058'   23                      INC     HL
0059'   C3 0050'                JP      MSGOUT
                        ;
                        ;------ 1 character out subroutine
005C'                   CHROUT:
005C'   CD 3E0D                 CALL    PCOUT
005F'   C9                      RET
                        ;
                        ;------ 1 character key input subroutine
0060'                   CHRIN:
0060'   CD 3583                 CALL    PCIN
0063'   CD 3E0D                 CALL    PCOUT
0066'   C9                      RET
                        ;
                        ;
                        ;------ string data and stack area ------
0067'   0D 0A 31 2E     MSGMNU: DB      CR,LF,'1. Morning'
006B'   20 4D 6F 72
006F'   6E 69 6E 67
0073'   0D 0A 32 2E             DB      CR,LF,'2. Noon'
0077'   20 4E 6F 6F
007B'   6E
007C'   0D 0A 33 2E             DB      CR,LF,'3. Night'
0080'   20 4E 69 67
0084'   68 74
0086'   0D 0A 34 2E             DB      CR,LF,'4. Bye'
008A'   20 42 79 65
008E'   0D 0A 0A 49     MSGINP: DB      CR,LF,LF,'Input 1,2,3 or 4  >'
0092'   6E 70 75 74                     ,EOS
0096'   20 31 2C 32
009A'   2C 33 20 6F
009E'   72 20 34 20
00A2'   20 3E 00
                        ;
00A5'   0D 0A 0A 2A     MSGMNG: DB      CR,LF,LF,'** Good morning **'
00A9'   2A 20 47 6F                     ,CR,LF,EOS
00AD'   6F 64 20 6D
00B1'   6F 72 6E 69
00B5'   6E 67 20 2A
00B9'   2A 0D 0A 00
00BD'   0D 0A 0A 2A     MSGNOO: DB      CR,LF,LF,'** Good afternoon **'
00C1'   2A 20 47 6F                     ,CR,LF,EOS
00C5'   6F 64 20 61
00C9'   66 74 65 72
00CD'   6E 6F 6F 6E
00D1'   20 2A 2A 0D
00D5'   0A 00
```

```
00D7'   0D 0A 0A 2A      MSGNIT: DB      CR,LF,LF,'** Good night **',CR┐
00DB'   2A 20 47 6F                                             └→,LF,EOS
00DF'   6F 64 20 6E
00E3'   69 67 68 74
00E7'   20 2A 2A 0D
00EB'   0A 00
                         ;
                                 END




        MACRO-80 3.44   09-Dec-81       PAGE    S


Macros:

Symbols:
0060'   CHRIN           005C'   CHROUT          000D    CR
0000    EOS             002B'   JOB1            0037'   JOB2
0043'   JOB3            004F'   JOB4            0006'   KEYIN
000A    LF              008E'   MSGINP          00A5'   MSGMNG
0067'   MSGMNU          00D7'   MSGNIT          00BD'   MSGNOO
0050'   MSGOUT          3583    PCIN            3E0D    PCOUT
0000'   START

No Fatal error(s)
```

では，このソース・プログラムのファイル名を「SELECTPC . MAC」として，アセンブルからロード，実行までの一連の実行例を示しましょう．

**図6-4-8 N88-BASIC用プログラムの開発実行例**

M80によるアセンブラの実行は，図6-4-1と同様なので，すでにアセンブル済とする

```
A>DIR SELECTPC.* ↵ ……………… アセンブル後の「SELECTPC」ファイルの確認

A: SELECTPC MAC : SELECTPC REL : SELECTPC PRN
                  生成されたリロケータブル・  生成されたアセンブル・
                  オブジェクト・ファイル      リスト・ファイル
A>L80 /P:D000,SELECTPC,SELECTPC/N/X/E ↵ …リンクローダの実行，スタート・アドレスをD000Hとして，インテルHEX形式のオブジェクトを生成する
Link-80  3.44  09-Dec-81  Copyright (c) 1981 Microsoft
                 エンド・アドレス
Data    D000    D0ED      <  237>
プログラムのスタート・アドレス    プログラム全体のバイト数
40781 Bytes Free
[0000   D0ED      208]
Origin above loader memory, move anyway(Y or N)?N
リンクローダ実行終了
A>DIR SELECTPC.* ↵ ……………… 「SELECTPC」ファイルの確認

A: SELECTPC MAC : SELECTPC REL : SELECTPC PRN : SELECTPC HEX

A>                       生成されたインテルHEX形式のオブジェクトファイル
```

```
A>TYPE SELECTPC.HEX ↵ ……… 生成されたD000HをスタートアドレスとするインテルHEX形式のオブジェクトをタイプアウトしてみる

:20D000002167D0CD50D0CD60D0FE31CA2BD0FE32CA37D0FE33CA43D0FE34CA4FD03E3FCD66
:20D020005CD0218ED0CD50D0C306D021A5D0CD50D0CD60D0C300D021BDD0CD50D0CD60D014
:20D04000C300D021D7D0CD50D0CD60D0C300D0FF7EB7C8E5CD5CD0E123C350D0CD0D3EC956
:20D06000CD8335CD0D3EC90D0A312E204D6F726E696E670D0A322E204E6F6F6E0D0A332E31
:20D08000204E6967687400D0A342E204279650D0A0A496E7075742031 2C322C33206F72205D
:20D0A0003420203E000D0A0A2A2A20476F6F64206D6F726E696E67202A2A0D0A000D0A0ADA
:20D0C0002A2A20476F6F642061667465726E6F6F6E202A2A0D0A000D0A0A2A2A20476F6F22
:0DD0E00064206E69676874202A2A0D0A001A
:00000001FF

A>  ロード・アドレス    オブジェクト・プログラム
```

生成されたCP/Mのディスク上のオブジェクト・プログラム「SELECTPC.HEX」を，N88-BASICのディスク上に持ってくることができれば，5.2章で紹介したDUAD-88Dを使っても，メモリ上にロードし，実行することが可能

```
+++ LOADER  Ver 1.0 +++
```
市販されているいくつかのCP/M→N88-BASICのファイル変換プログラムを使って，N88-BASICのディスク上に持ってきたとする

```
OFFSET ? 0 ↵ ································オフセットは必要なし
File name? 2:SELECT.hex ↵ ···ファイル名の字数の制限があるので，N88BASICのディスク上では
                              「SELECT.hex」としてある。「.hex」は小文字でなければならない
D000-D0EC ································プログラムがロードされたアドレス範囲
 D000-D0EC


Ok ··········································メモリ上に実行可能な純マシン語に変換されてロード終了
MON ↵ ·····································モニタに入る


h]DCFF0,D0FF ↵ ··························ロードされたプログラムをダンプして確認する
CFF0 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00
D000 21 67 D0 CD 50 D0 CD 60 D0 FE 31 CA 2B D0 FE 32   !gミヘPミヘ`ミ 1ハ+ミ 2
D010 CA 37 D0 FE 33 CA 43 D0 FE 34 CA 4F D0 3E 3F CD   ハ7ミ 3ハCミ 4ハOミ>?ヘ
D020 5C D0 21 8E D0 CD 50 D0 C3 06 D0 21 A5 D0 CD 50   ¥ミ!■ミヘPミテ ミ!・ミヘP
D030 D0 CD 60 D0 C3 00 D0 21 BD D0 CD 50 D0 CD 60 D0   ミヘ`ミテ ミ!スミヘPミヘ`ミ
D040 C3 00 D0 21 D7 D0 CD 50 D0 CD 60 D0 C3 00 D0 FF   テ ミ!ラミヘPミヘ`ミテ ミ
D050 7E B7 C8 E5 CD 5C D0 E1 23 C3 50 D0 CD 0D 3E C9   ~キ■■ヘ¥ミ■#テPミヘ >ノ
D060 CD 83 35 CD 0D 3E C9 0D 0A 31 2E 20 4D 6F 72 6E   ヘ■5ヘ >ノ  1. Morn
D070 69 6E 67 0D 0A 32 2E 20 4E 6F 6F 6E 0D 0A 33 2E   ing  2. Noon  3.
D080 20 4E 69 67 68 74 0D 0A 34 2E 20 42 79 65 0D 0A    Night  4. Bye
D090 0A 49 6E 70 75 74 20 31 2C 32 2C 33 20 6F 72 20    Input 1,2,3 or
D0A0 34 20 20 3E 00 0D 0A 0A 2A 2A 20 47 6F 6F 64 20   4  >   ** Good
D0B0 6D 6F 72 6E 69 6E 67 20 2A 2A 0D 0A 00 0D 0A 0A   morning **
D0C0 2A 2A 20 47 6F 6F 64 20 61 66 74 65 72 6E 6F 6F   ** Good afternoo
D0D0 6E 20 2A 2A 0D 0A 00 0D 0A 0A 2A 2A 20 47 6F 6F   n **     ** Goo
D0E0 64 20 6E 69 67 68 74 20 2A 2A 0D 0A 00 00 FF 00   d night **
D0F0 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00
h]GD000 ↵ ································アドレスD000Hからプログラムを実行
1. Morning
2. Noon
3. Night
4. Bye
```

```
X ……………………何らかのキーを入力
Input 1,2,3 or 4  >1

** Good morning **……
                    CP/M上のものとまったく同様に実行される
1. Morning
2. Noon
3. Night
4. Bye
X
Input 1,2,3 or 4  >3

1. Morning
2. Noon
3. Night
4. Bye
X
Input 1,2,3 or 4  >4 ……………………プログラムを終了させる「4」を入力
h] ……………………モニタに戻った
```

以上で，小さなプログラムのソフトウェア開発(プログラミングの構想段階から最終的なプログラムの完成，実行まで)の実習を終わります．

# 6/5 スタックエリアについて

スタックについては，『はじめて読むマシン語』でも詳しく解説しているとおり，アセンブラを扱うには必要不可欠の重要な概念です．スタックとは本来，CPU が CALL 命令を実行するとき，その戻り番地を記憶しておくために，CPU が内部的に使ったり，各レジスタの値を一時退避しておいたりするものです．*

コール命令が実行されるために必要なスタックエリアのバイト数は，ネスティング（入れ子：サブルーチンの中にまたサブルーチンがあり，その中にまた…，という状態）していなければ2バイト（1つのアドレスを表すためには2バイトが必要）ですが，ネスティングが深くなると「ネスティングの深さ×2」バイトが必要になります．また，PUSH 命令の実行にも同じく2バイトが必要です．CALL 命令および PUSH 命令により使われたスタックは，RET 命令および POP 命令の実行により，それぞれ2バイトずつ解放されます．通常のプログラムにはたいてい CALL 命令や RET 命令が使われていますので，メモリ上の適当な箇所にスタックエリアを設けなければならないわけです．

ただし CP／M や BASIC が実行中である場合は，それ自身が動作するために必要なスタックエリアがすでに設定されています．CP／M や BASIC 上からユーザー・プログラムを実行する場合は，それぞれが使っていたスタックエリアがそのままユーザー・プログラムに引き継がれます．よって，ユーザー・プログラムの中で新たにスタックエリアを設定しなくても，以前のものがそのまま使われますので，プログラムは支障なく動作するはずです．しかし，ユーザー・プログラムに引き継がれた時点での，使用可能なスタックエリアには限りがあり，CP／M の場合は7レベル（7重のネスティングまで可能）しかありません．

＊スタックをデータ構造のひとつの形式としてとらえることもある．しかし本書では，このようにCPUに密着した機能として解説している．

従って，これ以上のスタックエリアを必要とするユーザー・プログラムの場合は，ユーザー・プログラム自身でスタックエリアを設定しなければならないわけです．また，プログラムをROM化して制御機器に組み込み，完全にそれだけのプログラムで動作させるような場合にも，独自のスタックエリアの設定が必要です．

これらのことに関する状況を，次の図で示しておきましょう．

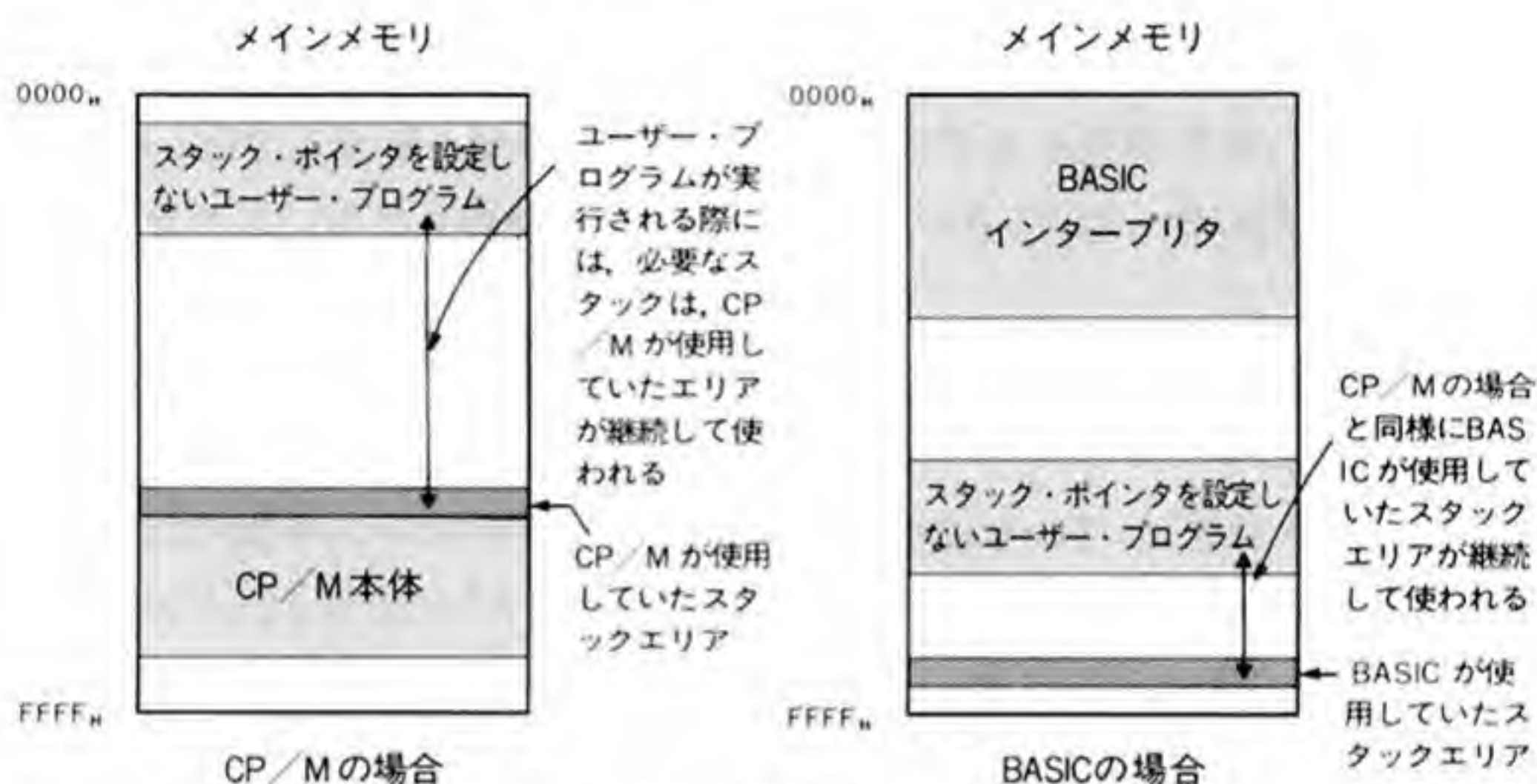



**図6-5-1　スタックエリアを設定しない場合**

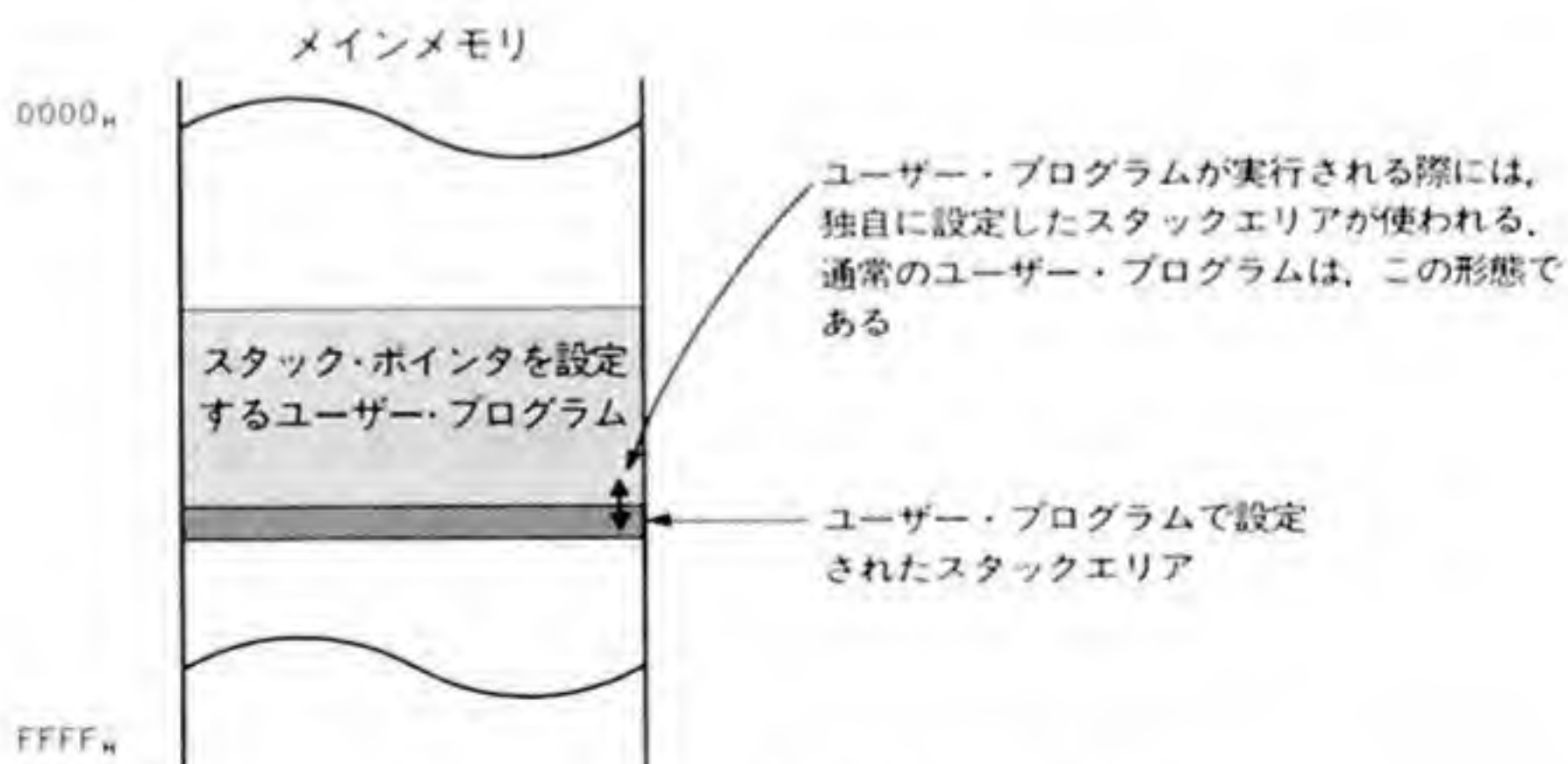



**図6-5-2　スタックエリアを設定した場合**

# 7 プログラミングの基本

プログラムは，アセンブラに限らずすべての言語において，構造的に書かれていなくてはなりません．確かに，思いつくままに書けるものから雑然と並べていっても，ソース・プログラムを記述することは可能です．また，そのプログラムに誤りさえなければ，アセンブルは可能ですし，完成したプログラムは目的どおりの働きをするでしょう．

しかし，ひとつのソフトウェアが完成するまでには，ソース・プログラムの修正，再アセンブル，実動テストなどによるデバッグ作業が，何回となく繰り返されます．また，完成してからも，ユーザー側の仕様の変更があるたびに，この一連の作業を繰り返さなければなりません．

このような状況で，ソース・プログラムが雑然としていて，読むことが困難なものであったらどうなるでしょう．おそらく，プログラムを作った本人でさえ整理ができなくなってしまい，お手上げの状態となることでしょう．またそれよりも，そのようなやり方では，小規模のものを除いて，ソース・プログラムそのものを組み立て得るかどうかも疑問です．

初心者であれ，プロフェッショナルであれ，プログラミングの基本は，

階層的構造を持つこと

であり，これが最も大切です．本章では，このようなプログラミングの基本について解説しましょう．

# 7/1 モジュール化と階層構造

「階層的構造」を持ったプログラムを書くというと，何かたいそう難しく高度な理論のように聞こえますが，そうではありません．誰もが当然と思うような，極めて常識的なことをいっているに過ぎないのです．

まず，「構造的」の意味ですが，これは，建築の場合を考えればよいでしょう．基礎を作り，骨格となる柱を組み，床や壁を張り，…というように，建造物はある構造を持って構築されます．

プログラムも，「物」でこそありませんが，建造物と同じように，構造を持って構築しなければなりません．つまり，ソフトウェアを開発するには，Z-80の命令語やアセンブラの書式や規則などに精通しているだけでなく，プログラム全体を見とおし，その構造を決定する能力が必要なのです．

プログラムに構造を持たせるには，「モジュール化」および「階層化」という2つの要素について考慮する必要があります．

モジュール化とは，あるプログラムを考える際に，そのプログラム全体をいくつかのグループやモジュール(ブロック)に機能別に分けることです．この具体例の簡単なものは，前章のメニュー選択のプログラムにも見ることができます．「EQU 定義部」，「メインルーチン部」，「分岐先の仕事部」，「サブルーチン部」……というように分割しているのがその一例です．

またこのようなモジュールを，ひとつのソース・プログラム内でなく，別々のソース・プログラムとして独立させ，それぞれをリロケータブル・アセンブラとリンクローダで開発し，最後に1本に結合したオブジェクト・プログラムを得る手法もあります．*

階層化とはモジュール化が行われていることを前提にした概念で，各モジュールをそれぞれの機能に従って構造的に積み上げることをいいます．つま

*この概要については，11.2章で解説する．

り、メインルーチン、サブルーチン、そのまたサブルーチン、そのまたサブルーチン、…というような、階層を成すわけです。このことを次の図で表してみましょう。

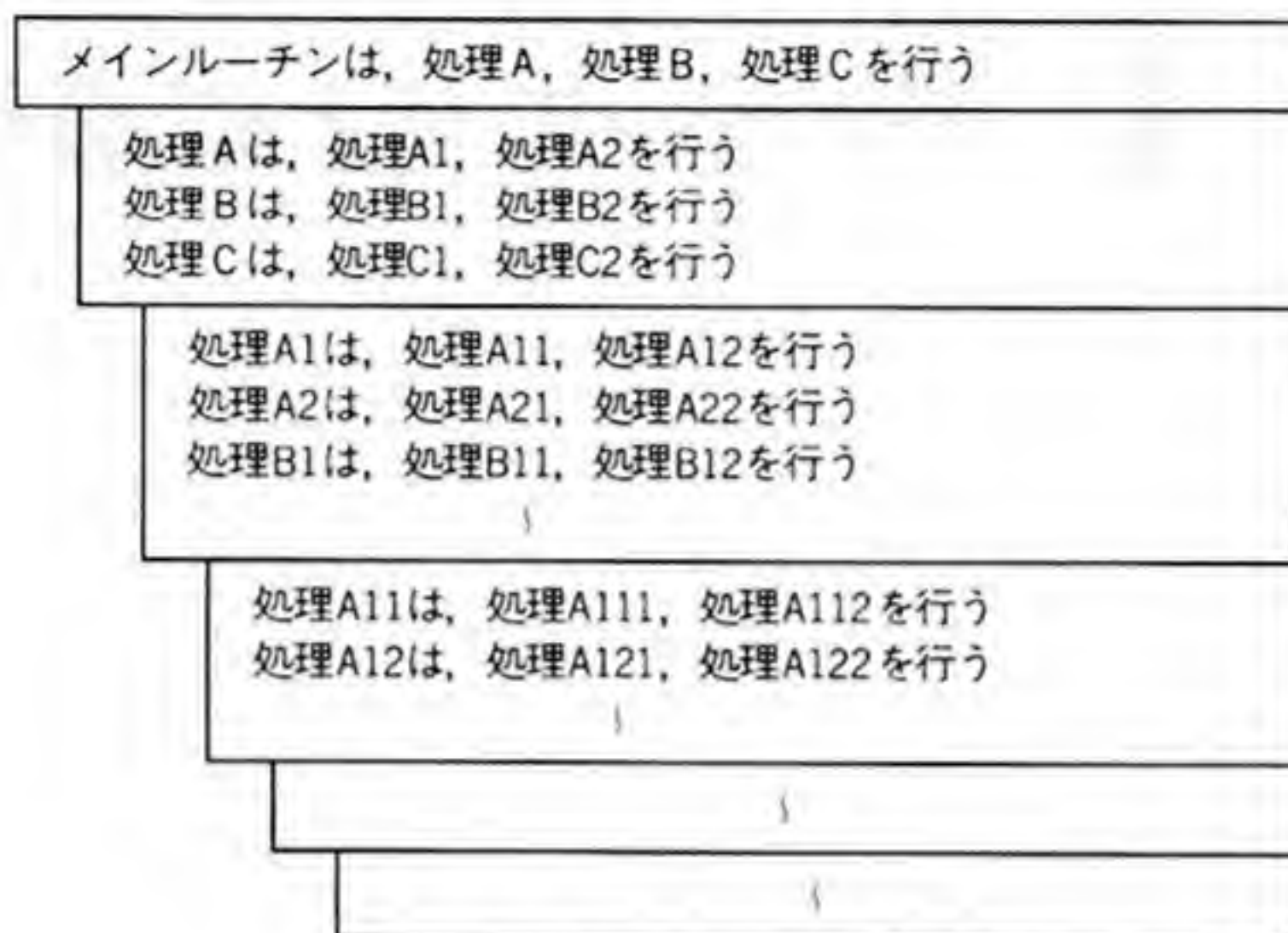



図7-1-1 プログラムの階層構造

これを、前章のプログラムの中の文字列出力サブルーチン「MSGOUT」を例に、次の図でもう少し具体的に示しましょう。

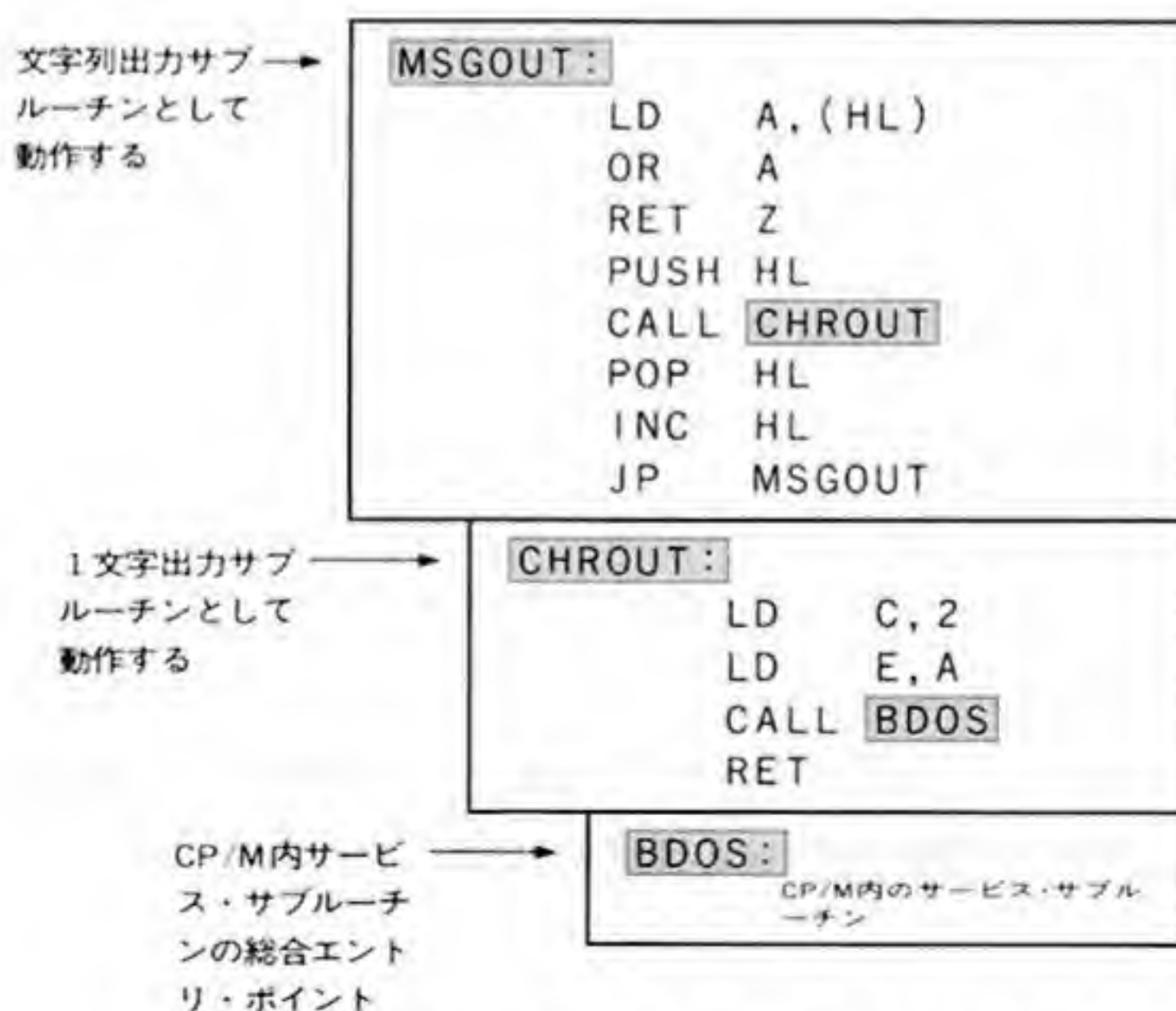



図7-1-2 文字列出力サブルーチンの階層構造

これは非常に簡単な例ですが，階層構造になっています．比較のために，階層構造をとらないプログラムを次に示します．

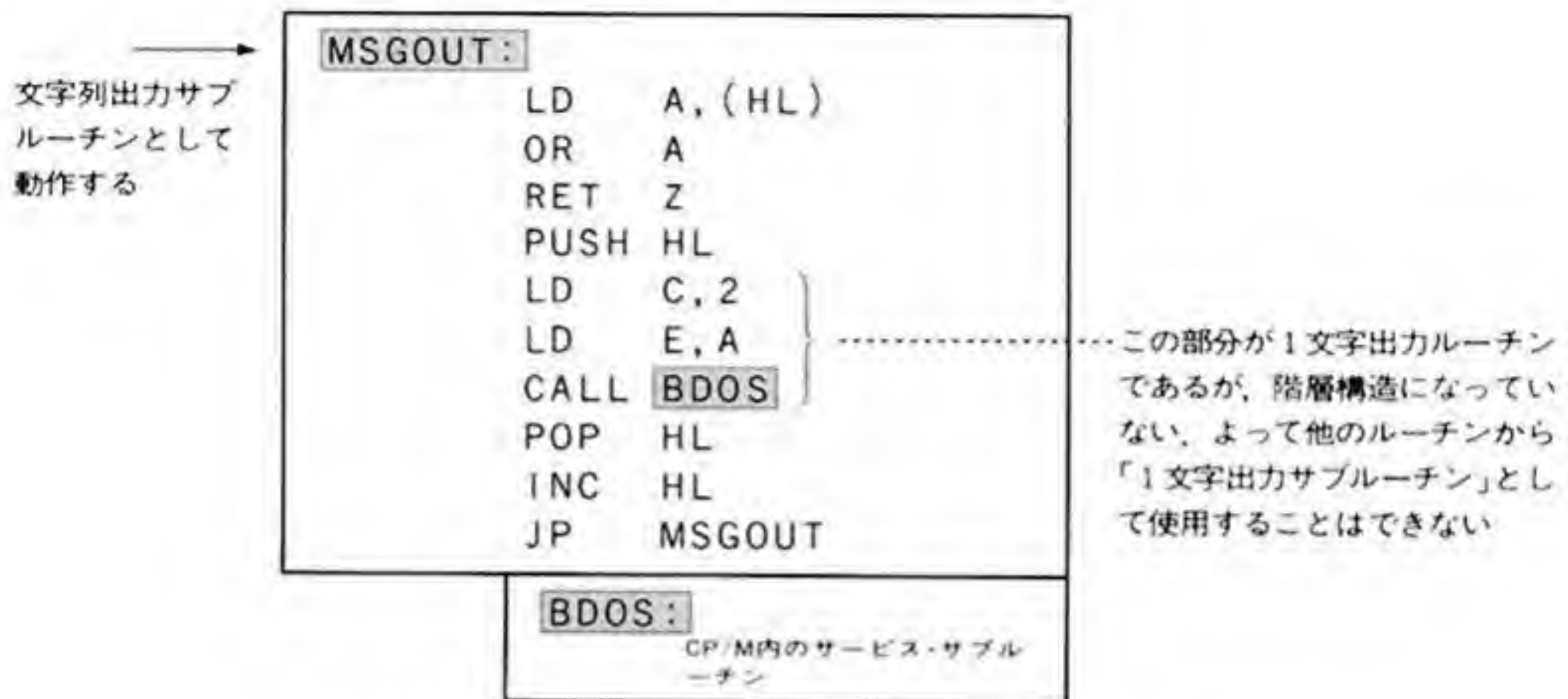



図7-1-3 階層構造をとらない文字列出力サブルーチン

このプログラムも,「CALL BDOS」というサブルーチンコールがあるので，厳密には階層構造になっているわけですが，1文字出力サブルーチン「CHROUT」はモジュール化されずに直接内部に組み込まれています．

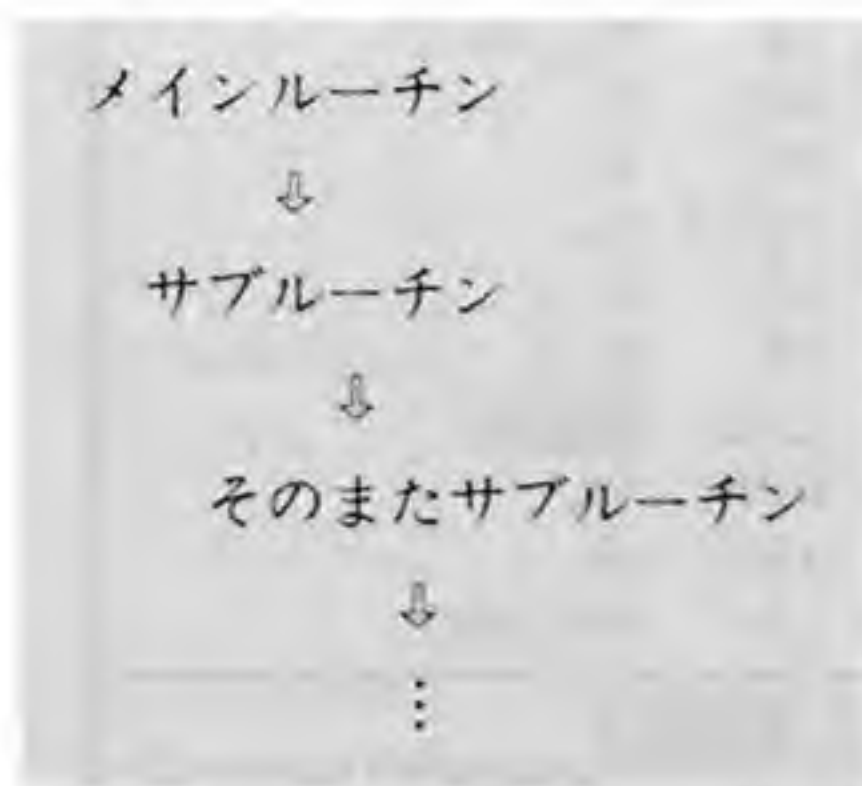
階層構造をとらないプログラムを階層構造を持つプログラムと比較すると，次の2つの主な相違点があります．

**(a) 文字列出力サブルーチン「MSGOUT」の構造が，多少わかりにくい**

例では，わかりやすさにそれほど大きな差はありません．しかし大きなプログラムになると，階層構造をとらない場合は，その構造の理解が難しくなります．

**(b) 1文字出力ルーチンが独立したモジュールではないので，他のモジュールからこれを利用することができない**

前章のプログラムはモジュール化されているため，別のモジュールで[？]を表示するときに，1文字出力サブルーチン「CHROUT」を使うことができました．モジュール化しないと，1文字出力ルーチンが必要なとき，それぞれの箇所で同じ1文字出力ルーチンを書かなければなりません．つまり，同じようなルーチンが重複して必要になります．

6．3章の各サブルーチンの項で「トップダウン」という言葉を使いましたが，アセンブラを階層構造で記述する場合は，通常トップダウンといって，上位モジュールから下位モジュールへ向かいます．つまり，

という構造になるわけです．

また，言語によってはトップダウンとは逆に，下位モジュールから上位モジュールに向かってのみ記述可能なものもあります．このことをトップダウンに対して，ボトムアップと呼んでいます．*

以上，簡単な例で解説しましたが，「階層的構造」プログラミングについて，その概要は理解できたのではないかと思います．

*6.3章でも最終的なリストはトップダウンの構造をもって記述されている．

# 7/2 アセンブリ・プログラムの基本的な形式

本書の例題プログラムの書き方などから，ほぼ見当がついているのではないかと思いますが，アセンブリ・ソース・プログラムを階層構造で記述する場合にどのような形式で記述するのか，その基本を解説しましょう．もっとも，ここでの例は，あくまで一般的で代表的なものですので，形式そのものにそれほどとらわれる必要はありません．単独のプログラムの場合の，代表的な階層構造の例が，次の図です．*

**図7-2-1　階層構造をもつ，一般的なプログラミングの形式**

*本章の7.1でも少し触れたが，1本の単独のプログラムの場合ではなく，ソース・プログラムを分割してそれぞれ独立に開発し，最後にリンクローダで1本のプログラムにまとめる「モジュール別ソフト開発法」(11章参照)の場合も，考え方は同じである．

これがプログラムの一般的な形式です．前章の**図6－3－2**のソース・プログラムをこれにあてはめてみてください．前章の例では，「システム初期設定部」は必要ないため（マシンが起動したときや，その上でCP/Mが起ち上がったときなどに，初期設定はすでに行われている），この部分は記述されていませんが，その他は，この図のようになっていることがわかると思います．

プログラムを階層構造にするといっても，その形式はただひとつというわけではありません．ここに示したものは，代表例にすぎませんので，これを参考に，それぞれのケースに適した構成にすればよいでしょう．また，本書のこれ以降に登場するプログラム例や他の参考書の例などを，本章で解説した「階層構造」の観点から，よく比較してみるとよいでしょう．

# 8 アセンブラの諸機能 実習解説

これまでの章で，アセンブラの最も基礎的な知識を解説してきました．本章では，これらの知識を基に，実用的なプログラムを作成する場合に必要となる，アセンブラのさらに多くの機能や使い方を，具体例を挙げて解説します．

アセンブラによるソフトウェア開発を能率よく行うには，まずZ-80や8080のCPU命令に関するプログラミングに精通する必要があります．またそれだけではなく，それらのプログラミングを側面から支援するために，アセンブラ自身が用意している多くの擬似命令やその他の機能をよく理解して，自由に使いこなせなければなりません．

本書をここまで読み進めば，アセンブラのプログラムには，CPU命令に関するものと，アセンブラ自身の機能（作成されるプログラムの働きのことではなく，アセンブラそのものの働き）をコントロールするものとの2つの要素があることは，すでに理解されているでしょう．

本書は，2つの要素のうち，アセンブラ自身の機能やその使い方を中心に解説することを目的としていますので，この章ではそれらの主なものを，具体的な実行例で解説しましょう．ここで解説する機能や使い方は，実務用のアセンブラには必ず備えられている基本的なものであり，実行例には，CP/Mの「ASM」，およびマイクロソフト社の「M80」アセンブラを使っています．

# 8.1 数値

アセンブラのアーギュメント・フィールド*では，2進数，8進数，10進数，16進数のうち，任意の形式の数値を使うことができ，それぞれを混用してもかまいません．

普通は，10進数がデフォルト（標準状態）ですので，ただ"1000"と書いた場合は，10進数の1000を表します．また，最後にDをつけて，"1000D"と書いた場合も，10進数の1000を表します．

これに対して，2，8，16進数を表す場合は，それぞれの数字の後に，B，O，Hの文字をつけて区別します．8進数を表す「O」は，ゼロと混同しやすいので，「O」の代わりに「Q」を使うアセンブラもあります．

- B　2進数　Binary
- O　8進数　Octal
- D　10進数　Decimal
- H　16進数　Hexadecimal

ソース・プログラム上のオペランドには，マイナス記号をつけた数値も記述できますが，オブジェクトデータとしては，符号なしの16ビットとして扱われます．よって，表せる数値の範囲は，0～$FFFF_H$，10進数では，0～65535となります．**

これらの書き方の一例を次のページに表で示しましょう．

当然のことながら，2進数では0と1，8進数では0～7，16進数では0～Fの数字を使います．2進数を使うと，値をビットパターンで表現できるので，アセンブラを扱う上ではたいへん便利なこともあるでしょう．例えば，ビット2とビット4を1に，他を0にして，ポートからデータを出力する場合など，ソース・プログラム上で，その値をビットパターンのまま，"00010100B"

*オペランドともいう．3.1章参照．
**詳しくは本章の8.2で解説する．

| 数値表現の形式 | 10進数の10000をそれぞれの形式で表す |
|---|---|
| 2進数 | 0010011100010000B |
| 8進数 | 23420O |
| 10進数 | 10000 |
| | 10000D |
| 16進数 | 2710H |

| | | | | | |
|---|---|---|---|---|---|
| 2進数 → | 0010 | 0111 | 0001 | 0000 | B |
| 16進数 → | 2 | 7 | 1 | 0 | H |

← それぞれ4ビットごとに区切ると16進数と対比しやすい

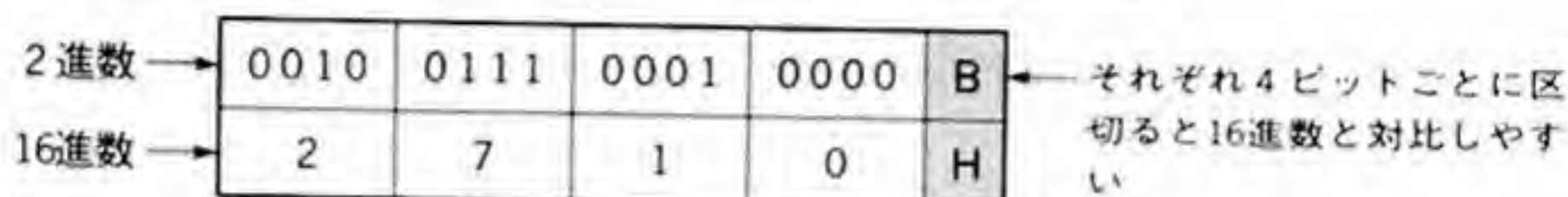



図8-1-1　2,8,10,16進数の表し方

と記述できます．

16 進数を扱う場合に注意することは，A～Fで始まる値は，「数字」のA～Fであることをアセンブラに認識させるために，その先頭に0を付加しなければならないということです．例えば，C000Hを示すには，ソース・プログラム上では"0C000$_H$"と書きます．

では次に，これらのいろいろな記述例のソース・プログラムを作り，実際にアセンブルして，そのアセンブルリストで確認してみましょう．アーギュメント・フィールド(オペランド)に記述したそれぞれの数値と，アセンブルされることによって生成された，16進数で表されている左端のオブジェクト欄の値とを，よく対比してください．

図8-1-2　2,8,10,16進数の使用例

```
0008 =          gg      EQU     1000B ··················· 2進数
0200 =          hh      EQU     1000O ··················· 8進数
03E8 =          ii      EQU     1000  ··················· 10進数
1000 =          jj      EQU     1000H ··················· 16進数
                :
FFFF =          kk      EQU     1111111111111111B ········· 2進数
FFFF =          ll      EQU     777777O ·········· 8進数
FFFF =          mm      EQU     65535   ·········· 10進数
FFFF =          nn      EQU     0FFFFH  ·········· 16進数
                :
4500 =          A1      EQU     4000H + 500H ········ 16進数+16進数
41F4 =          A2      EQU     4000H + 500  ········ 16進数+10進数
4140 =          A3      EQU     4000H + 500O ········ 16進数+8進数
AB80 =          A4      EQU     1010101100000000B + 128 ········ 2進数+10進数
```

それぞれの行の右側の数値がアセンブルされて，ここに16進数で表示されている

適当につけたシンボル

それぞれの形式を表す末尾のB，O，Hに注意

# 8/2 演算子

アーギュメント・フィールドには数値のみでなく，「演算子」(オペレータとも呼ばれる)を用いた式を使用することができます．演算子には，通常の加減乗除などを行う算術演算子と，AND, OR, シフトなどを行う論理演算子とがありますが，ここではそれらの代表的なものについて解説しましょう．

aおよびbが，シンボル，ラベル，定数，あるいは1～2文字の文字列などの場合，演算子を使って，「a＋b」,「a−b」あるいは，「a AND b」などの記述が可能であり，これらの演算は，アセンブラの内部で行われます．

前節でも述べましたが，すべての計算は，符号なしの16ビットで行われますので，算術演算では，例えば，

$$\mathrm{FFFF_H} + 1 = 0$$

⇩ つまり

| | | | | | | |
|---|---|---|---|---|---|---|
| | 1111 | 1111 | 1111 | $1111_B$ | ……… | $\mathrm{FFFF_H}$ |
| ＋ | 0000 | 0000 | 0000 | $0001_B$ | ……… | 0001 |
| (1) | 0000 | 0000 | 0000 | $0000_B$ | ……… | 0000 |

↑

この桁上がりの17ビット目は無視される

となります．

もう一例を示すと，

$2-4=FFFE_H$ ……内部では $FFFE_H$ であることに注意

⇩ つまり

| | | | | | | |
|---|---|---|---|---|---|---|
| (1) | 0000 | 0000 | 0000 | $0010_B$ | ……… | 0002 |
| − | 0000 | 0000 | 0000 | $0100_B$ | ……… | 0004 |
| (−) | 1111 | 1111 | 1111 | $1110_B$ | ……… | $FFFE_H$ |

↑
桁借りとなるが17ビット目は
無視される

となります．ソース・プログラム上ではマイナス記号を使うことができますが，あくまで16ビットで表される範囲で扱われます．*

また，論理演算では，ビット単位に論理的に演算されますので，例えば，

$F0AA_H$ AND $AAF0_H = A0A0_H$

⇩ つまり

| | | | | | | |
|---|---|---|---|---|---|---|
| | 1111 | 0000 | 1010 | $1010_B$ | ……… | $F0AA_H$ |
| AND | 1010 | 1010 | 1111 | $0000_B$ | ……… | $AAF0_H$ |
| | 1010 | 0000 | 1010 | $0000_B$ | ……… | $A0A0_H$ |

となります．

＊アセンブラでマイナス表記を行う場合は，「FFFEH」を「−2」であると解釈できるような規則に従う．この規則を「2の補数表記」と呼ぶが，本書では詳しく解説しない．なお，2の補数表記に従うと16ビットで表現できる数値の範囲は−32768〜32767である．

次の表に，アセンブラで使われる代表的な演算子とその意味を示します．

| 演算子 | そ の 意 味 |
|---|---|
| +a | aと同じ |
| −a | 0−aと同じ(例えば−2＝FFFE$_H$となる．要注意) |
| a + b | 加算 |
| a − b | 減算 |
| a ＊ b | 乗算 |
| a / b | 除算 |
| a MOD b | a/bの余り |
| NOT a | aの各ビット(16ビット)をすべて反転する(1→0，0→1) |
| a AND b | 論理積a∧b(ビット単位) |
| a OR b | 論理和a∨b(ビット単位) |
| a XOR b | 排他的論理和a$\veebar$b(ビット単位) |
| a SHL b | aの各16ビットをbだけ左にシフトする |
| a SHR b | aの各16ビットをbだけ右にシフトする |

図8-2-1　代表的な演算子とその意味

では，これらの演算子を使い，いろいろな記述例のソース・プログラムを作成し，実際にアセンブルして確認してみましょう．演算結果が示されているアセンブルリストのオブジェクトコード部の値に注目してください．

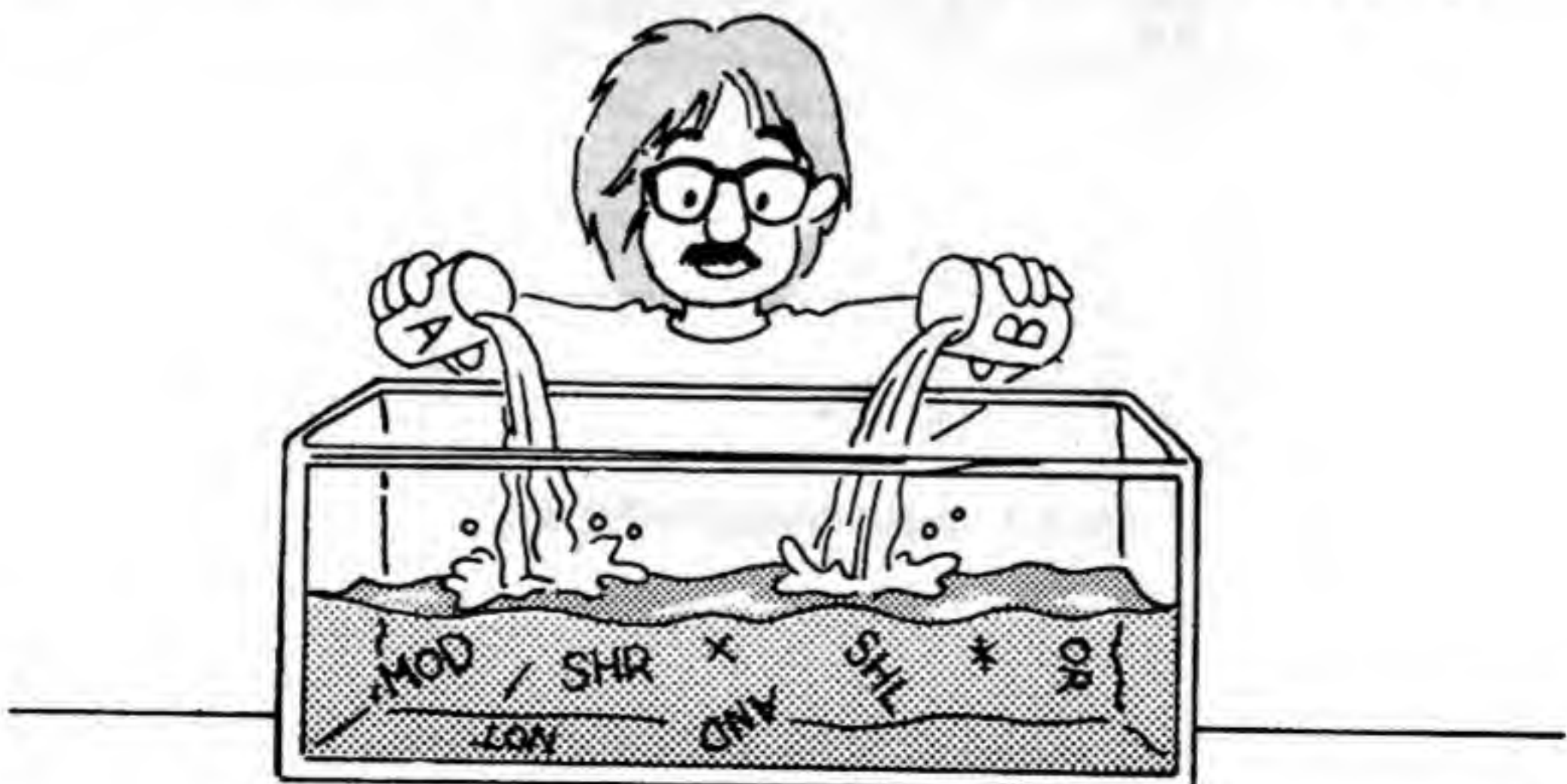



図8-2-2　代表的な演算子の使用例

```
0002 =          gg      EQU     2
0004 =          hh      EQU     4
0005 =          ii      EQU     5
0019 =          jj      EQU     25
0064 =          kk      EQU     100
F0AA =          ll      EQU     1111000010101010B
AAF0 =          mm      EQU     1010101011110000B
                ;
0009 =          A1      EQU     hh + ii
0009 =          A2      EQU     4 + 5
C400 =          A3      EQU     0C000H + 400H
0003 =          A4      EQU     ii - gg
FFFF =          A5      EQU     hh - ii
6800 =          A6      EQU     9000H - 2800H
FFFE =          A7      EQU     -gg
0008 =          A8      EQU     2 * 4
09C4 =          A9      EQU     kk * jj
0004 =          A10     EQU     kk / jj
0000 =          A11     EQU     ii / jj
0000 =          A12     EQU     25 MOD 5
0001 =          A13     EQU     jj MOD hh
0002 =          A14     EQU     10 MOD 4
0F55 =          A15     EQU     NOT ll
FFFF =          A16     EQU     NOT 0
A0A0 =          A17     EQU     ll AND mm
A0A0 =          A18     EQU     0F0AAH AND 0AAF0H
FAFA =          A19     EQU     ll OR mm
5A5A =          A20     EQU     ll XOR mm
AA00 =          A21     EQU     ll SHL 8
0F0A =          A22     EQU     ll SHR 4
```



これらの演算子の間には，次の図に示す優先順位があります．

| 優先順位 | 演算子(←→方向は同順位) |
|---|---|
| 高 ↑ | * / MOD SHL SHR |
| | + − |
| | NOT |
| | AND |
| ↓ 低 | OR XOR |

図8-2-3　演算子の相互間の優先順位

( )を使うと，一般的な数学の計算の約束と同じように，上に示した順位には関係なく，( )内が最優先に計算されます．また，同じ行に同じ優先順位のものがある場合は，左から右に向かって順に計算が行われるのも数学の計算と同じです．

図8-2-4　計算式の優先順位の使用例

```
0002 =          gg     EQU    2                      |
0004 =          hh     EQU    4                      |
0005 =          ii     EQU    5                      | 以下の演算で使用する
F0AA =          ll     EQU    1111000010101010B      | シンボルの値の設定
AAF0 =          mm     EQU    1010101011110000B      |
0000 =          nn     EQU    0                      |
                 :
000E =          A1     EQU    hh + ii * gg
000E =          A2     EQU    4 + (5*2)
0012 =          A3     EQU    (hh+ii)*gg
000A =          A4     EQU    hh * ii / gg
0009 =          A5     EQU    3 * 4 - 6 / 2
0007 =          A6     EQU    5 + 10 MOD 4
F0AA =          A7     EQU    ll OR mm AND NOT mm + hh * gg
F0AA =          A8     EQU    ll OR (mm AND (NOT (mm + (hh * gg))))
結果を示すオブジェクト部
```

演算子を記述する際に注意すべきことのは，+，-，*，/，を除くすべての演算子は，1つ以上のスペースを空けて，前後の文字と分離する必要があるということです．ただし，( )で囲む場合は，( )そのものが分離記号になりますので，スペースを空けなくてもかまいません．これらのことを次の実例で示しましょう．アセンブル・エラーとなった例も示しています．

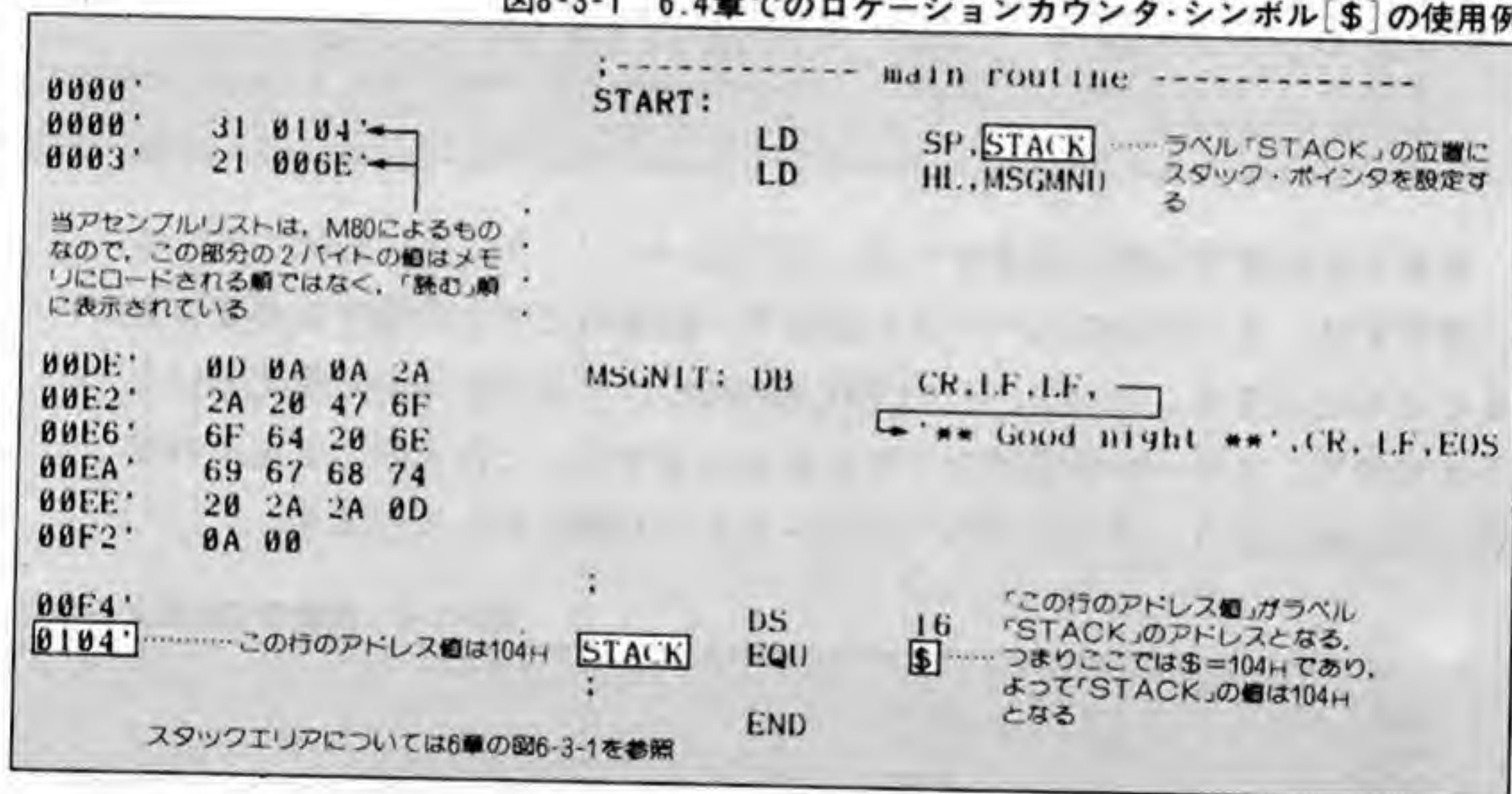
図8-2-5　演算子の記述上の制約

```
0002 =          gg     EQU    2                      |
0004 =          hh     EQU    4                      |
0005 =          ii     EQU    5                      | 前リストと同じシンボルの設定
F0AA =          ll     EQU    1111000010101010B      |
AAF0 =          mm     EQU    1010101011110000B      |
0000 =          nn     EQU    0                      |
                 :
000E =          A1     EQU    hh+ii*      gg
000E =          A2     EQU    4+(5*     2       )
0007 =          A3     EQU    5+10     MOD    4
U0000 =         A4     EQU    5+10MOD4 ·········10 MOD 4とすること
A0A0 =          A5     EQU    ll        AND mm
U0000 =         A6     EQU    llANDmm ··········ll AND mmとすること
A0A0 =          A7     EQU    (ll)AND(mm)
エラー表示
                              エラー表示のある2つの行を除いて，あとは
                              すべて正しい記述
```

# 8/3 ロケーションカウンタシンボル

ソース・プログラム中の任意の命令が位置するロケーション・カウンタの値(その命令のオブジェクト・プログラム上のアドレス値)を，[$]記号によって，アーギュメント・フィールド(オペランド)に記述することができます．6章の図**6-4-2**のリストの中で，スタック・ポインタの設定に使っているのがその一例です．その部分をもう一度示しましょう．

**図8-3-1　6.4章でのロケーションカウンタ・シンボル[$]の使用例**

```
                          ;------------- main routine -------------
0000'                     START:
0000'   31 0104'                  LD      SP,STACK
0003'   21 006E'                  LD      HL,MSGMNI)
                          :
                          :
                          :
                          :
00DE'   0D 0A 0A 2A       MSGNIT: DB      CR,LF,LF,
00E2'   2A 20 47 6F                       '** Good night **',CR,LF,EOS
00E6'   6F 64 20 6E
00EA'   69 67 68 74
00EE'   20 2A 2A 0D
00F2'   0A 00
                          :
00F4'                             DS      16
0104'                     STACK   EQU     $
                          :
                                  END
```



このような[$]記号は，ロケーションカウンタ・シンボルと呼ばれ，「ここのアドレス値」という意味を持っています．つまり，[$]を「ここのアドレス値」と読み換えればよいわけです．

「ロケーション」とは，ソース・プログラムのそれぞれの命令に対する，オブジェクト・プログラム上でのアドレスのことです．また，「ロケーション・カウンタ」とは，アセンブラが，ソース・プログラムの先頭から順にアセンブルしながらオブジェクトコードを生成していく際，次にアセンブルする命

令に対するアドレスを保持しているカウンタのことです.

このカウンタは,「プログラム・カウンタ」とも呼ばれます. これはアセンブラの中にソフトウェア的に存在するカウンタですので,CPUのレジスタの1つである「PC」(プログラム・カウンタ)とはまったく別物ですが, その機能はよく似ています. 前者は次にアセンブルする命令(ニーモニック)が置かれているアドレスを保持し,後者は次に実行する命令コード(オブジェクトコード)が置かれているアドレスを保持しています.

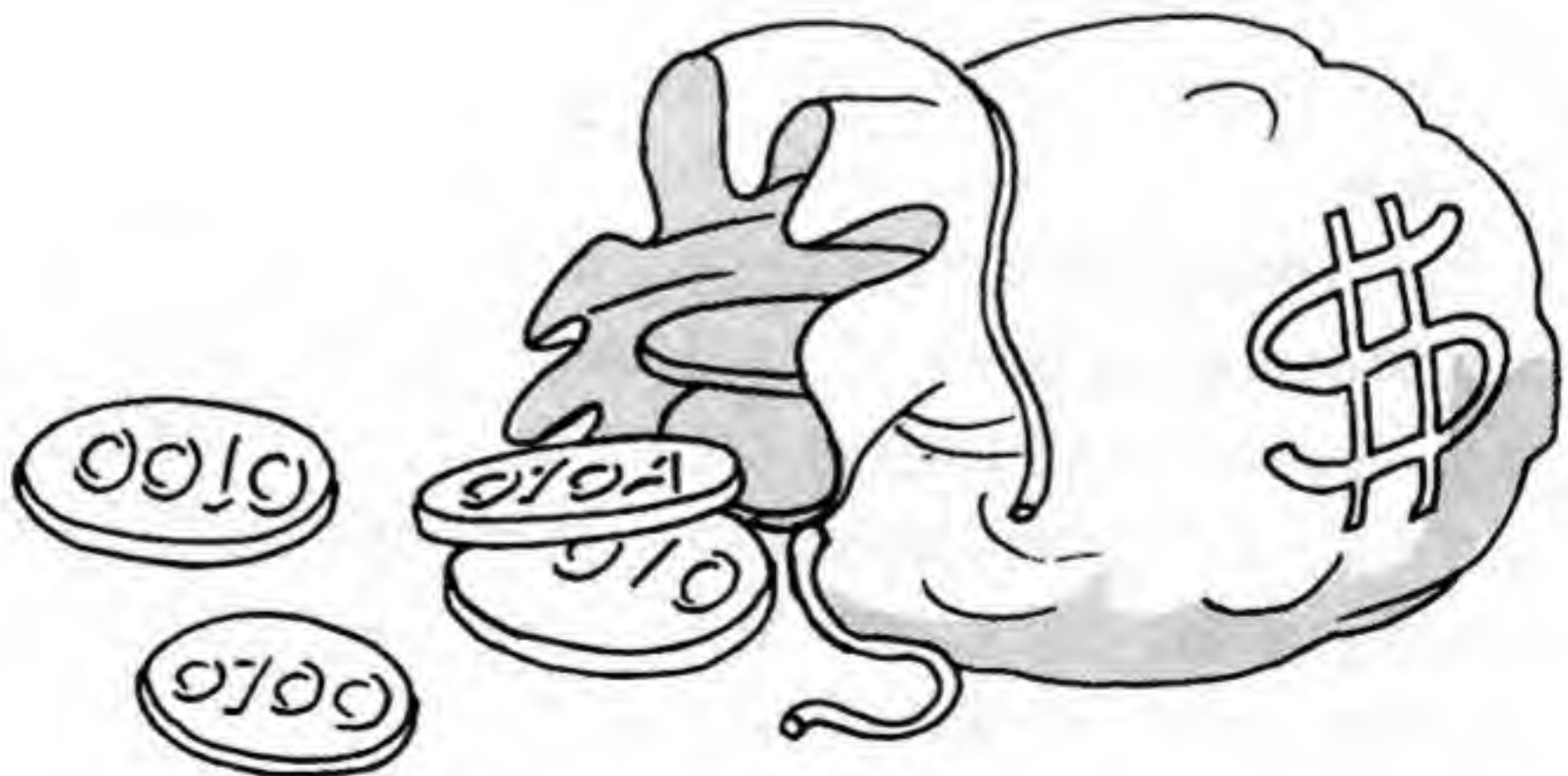
このようなことから, ロケーションカウンタ・シンボル[$]は,「現在のプログラム・カウンタの値を代用する記号」という意味で,「カレント・プログラムカウンタ・シンボル」とも呼ばれます. この使い方の実例を, もう少し示しておきましょう.

図8-3-2 ロケーションカウンタ・シンボル[$]の使い方

```
                              ORG     100H
                        ;
0100   1E58                   LD      E,'X'
0102   0E02                   LD      C,2
0104   CD0500                 CALL    0005H
0107   C30001                 JP      $-7
                        ;
010A   0A01             ABUF: DW      $
010C   (0000)                 END
```

JP $-7：この行のアドレス値は107H, よって$-7は107H－7H＝100Hであり, このジャンプ命令はアドレス100Hにジャンプする

DW $：この行のアドレス値は10AHであり, $＝010AHとなる. よってDW擬似命令(後述)で0A 01の2バイトがプログラム中に取り込まれる

当リストのアセンブラは, 前リストのアセンブラと異なるものを使用しているので, メモリにロードされる順に表示されている

# 8/4 擬似命令

擬似命令については，2章でORG, END, EQU, DBなどの，最も基本的なものを取り上げ，例題プログラムに使用して予備知識的な解説をしました．

本節では，さらにいくつかの基本的な擬似命令を加えて，それらを機能別にグループに分け，擬似命令のいろいろな使い方について実習解説しましょう．

## ロケーションカウンタ指定

### * ORG （ORiGin： オリジン指定）

ソース・プログラム中にORG擬似命令が現れた時点で，ロケーション・カウンタは，この命令のアーギュメント・フィールドに指定されている値(絶対値でも，定義済みのシンボルでもよい)をそのアドレスとしてセットします．つまり，ORG擬似命令の次の命令からは，そこで指定されたアドレスを先頭にしてオブジェクトコードに変換されるわけです．

この擬似命令は，ひとつのソース・プログラムの中で複数の使用が可能ですが，そのために生成されるオブジェクト・プログラムによって使われるメモリエリアが重複しても，それをアセンブル・エラーとして検出する機能はありません．よって，複数のORG擬似命令を使用する場合には，メモリエリアの重複については注意が必要です．

では，ORG擬似命令の使用例を示しましょう．このプログラムは，アドレス $0100_H$に置かれる部分と，アドレス $4000_H$に置かれる部分との2つでできており，その2つを交互に実行するものです．この2つの部分は，ソース・プログラムの2つのORG擬似命令によって切り分けられています．このプログラム名を「8ORG」として，CP/M上でアセンブルした後のアセンブルリストと，DDTでの2，3の実行例を示しましょう．

**図8-4-1　ORG擬似命令の使用例**

```
        (0100)      ADRTOP EQU      100H
                    ;                ┌──このようにシンボルを使って指示してもよい
                           ORG      ADRTOP
                    ;
┌[0100]  1E58       START1:         LD       E,'X'  アドレス100H からのプログラム.CRT
│ 0102   0E02              LD       C,2             ディスプレイに文字「X」を表示し，アド
│ 0104   CD0500            CALL     0005H           レス4000H にジャンプする
│ 0107   C30040            JP       START2
│ ....2カ所のロード・アドレス ;
│    に注目                 ORG      4000H
│                   ;
└[4000]  1E5A       START2:         LD       E,'Z'  アドレス4000H からのプログラム，文字「Z」
  4002   0E02              LD       C,2             を表示し，アドレス100H にジャンプする
  4004   CD0500            CALL     0005H
  4007   C30001            JP       START1
                    ;
  400A  (0000)             END
```

```
A>TYPE 8ORG.HEX ↵ ..........生成されているこのプログラム(「8ORG.HEX」)のHEX形式の
                            オブジェクト・プログラムをタイプアウトして確認する
:0A[0100]00[1E580E02CD0500C30040]9A ..............この行は0100H からのプログラム
:0A[4000]00[1E5A0E02CD0500C30001]98 ..............この行は4000H からのプログラム
:0000000000
     ↑                  ↑
A>                  オブジェクト部
  ロード・アドレス部
```

```
A>DDT 8ORG.HEX ↵ ..........HEX形式のオブジェクト・プログラム「8ORG.HEX」を，DDTでメモリ上に
                           実行可能なオブジェクト・プログラムに変換してロードする
DDT VERS 2.2
NEXT  PC
400A 0000
ロード終了
-D100 10F ↵ ..................0100H 付近のダンプ
0100 [1E 58 0E 02 CD 05 00 C3 00 40] 52 49 47 48 54 20 .X.......@RIGHT
                                    ↖
-D4000 400F ↵ .................4000H 付近のダンプ   ロードされた純マシン語のプログラム
                                    ↙
4000 [1E 5A 0E 02 CD 05 00 C3 00 01] 00 00 00 00 00 00 .Z..............
-G100 ..............0100H から実行，XとZが交互に表示される
XZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZ
ZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZX
XZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZ
ZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZX
XZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZ
ZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZX
XZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZXZ
実行を終了させるにはリセット・ボタンを押すこと            ・
                                                          ・
                                                          ・
```

このほかにも，ロケーション・カウンタに関する擬似命令には，リロケータブル・アセンブラで使用するASEG，CSEG，DSEG，COMMONなどがありますが，ここでは触れません.*

## シンボル定義

### * EQU (EQUate： シンボル定義)

EQU擬似命令は，シンボル・フィールドに書かれたシンボルに，アーギュメント・フィールドに書かれた値(絶対値，あるいはすでに定義済みのシンボル)を割りあてます．一度定義されたシンボルは，同じプログラム内で再定義することはできませんし，ラベルとして使うこともできません．もし別の値に定義し直す必要のある場合は，EQU擬似命令を使わずに，最初から次に説明する「SET」あるいは「ASET」擬似命令を使います．

### * SETまたはASET (再定義可能シンボル定義)

「SET」(または「ASET」)擬似命令は，シンボル・フィールドに書かれたシンボルに，アーギュメント・フィールドに書かれた値を割りあてます．ここまでは，「EQU」擬似命令と同じ機能ですが，「SET」や「ASET」で定義されたシンボルは，何度でも定義し直すことができます．つまりこれらは，再定義が可能なEQU擬似命令とも考えられます．

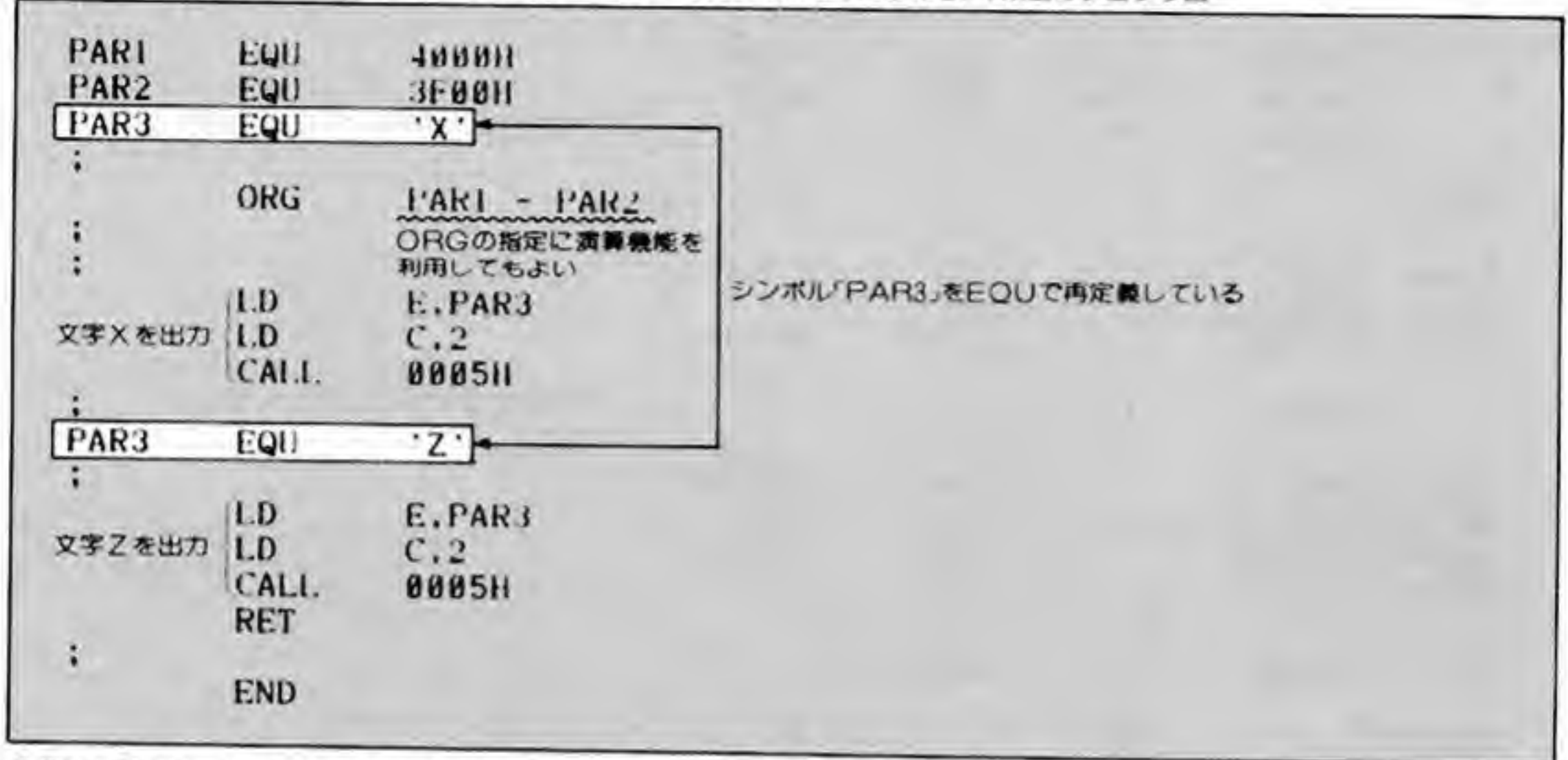
「SET」と「ASET」は，まったく同じ機能の擬似命令ですが，Z-80のインストラクション(CPU命令)のビットセット命令のニーモニックが，同じ「SET」であるため，マイクロソフト社のM80アセンブラなどでは，「ASET」が用意されています．Z-80のアセンブル時には，この「ASET」擬似命令を使用します．

では，前項のEQUを含めて，SET，ASET擬似命令のいくつかの使用例を示しましょう．

*ASEGおよびCSEGについては11章を参照．

**図8-4-2　EQU, SET, ASET擬似命令の使用例**

EQUを再定義したプログラム．XとZの2文字を表示するだけの簡単な内容のCP/M上のプログラム

```
PAR1        EQU     4000H
PAR2        EQU     3F00H
PAR3        EQU     'X'
;
            ORG     PAR1 - PAR2
;                   ORGの指定に演算機能を
;                   利用してもよい
           (LD      E,PAR3
文字Xを出力 LD      C,2
           (CALL    0005H
;
PAR3        EQU     'Z'
;
           (LD      E,PAR3
文字Zを出力 LD      C,2
           (CALL    0005H
            RET
;
            END
```



EQUを再定義することはできないので，このソース・プログラムをアセンブルすると，
アセンブル・エラーとなるはずである

そのアセンブル結果を示すアセンブルリスト．□部はエラー行

```
        (4000)      PAR1    EQU     4000H
        (3F00)      PAR2    EQU     3F00H
        (0058)      PAR3    EQU     'X'
                    *** multiple definition ***
                    ;
0000                        ORG     PAR1 - PAR2
                    ;
                    ;
0100    1E5A#               LD      E,PAR3
                    *** multiple definition ***
0102    0E02                LD      C,2
0104    CD0500              CALL    0005H
                    ;
        (005A)      PAR3    EQU     'Z'
                    *** multiple definition ***
                    ;
0107    1E5A#               LD      E,PAR3
                    *** multiple definition ***
0109    0E02                LD      C,2
010B    CD0500              CALL    0005H
010E    C9                  RET
                    ;
010F    (0000)              END

Errors              4
```

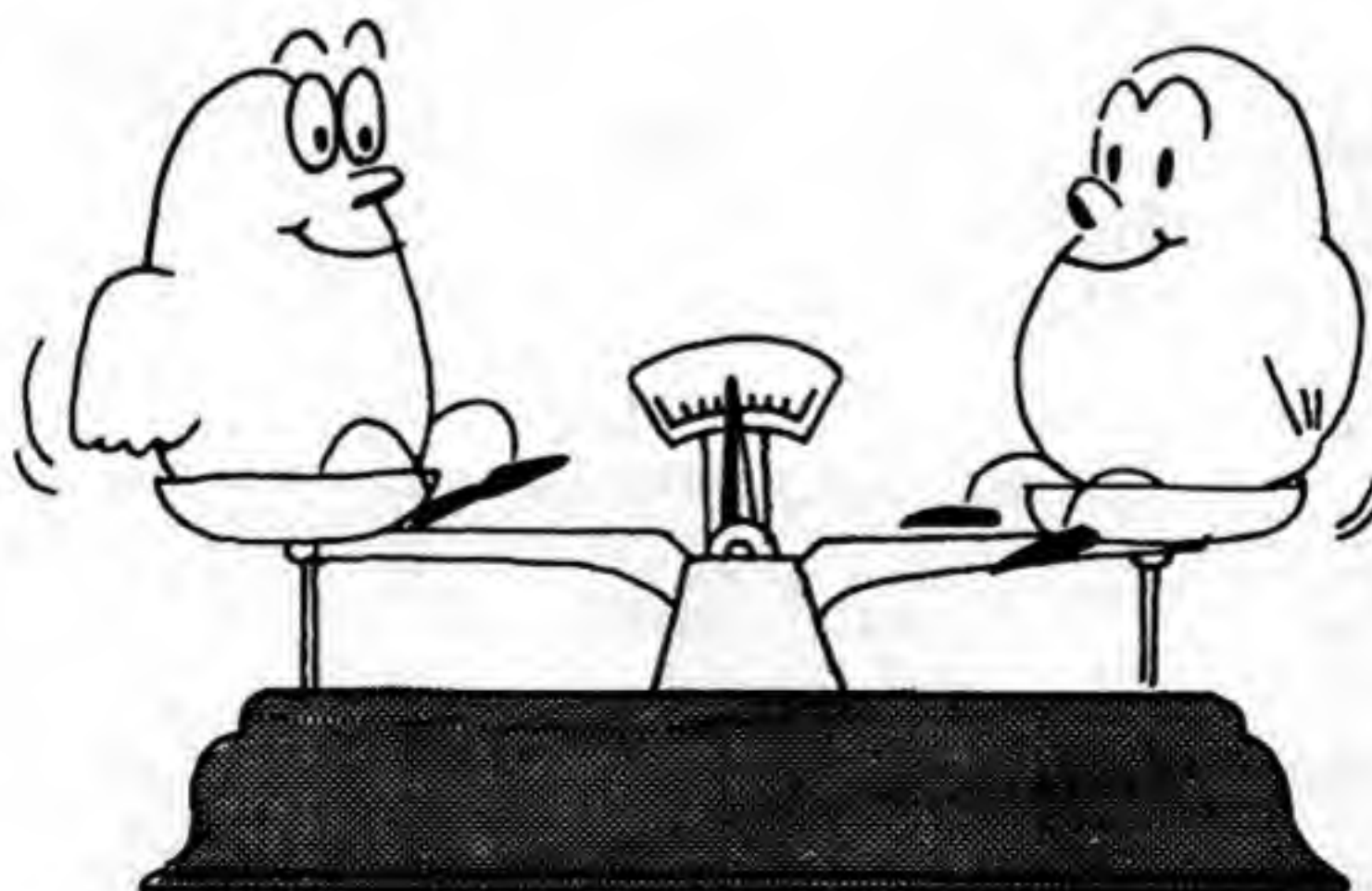
エラーあり

シンボル「PAR3」に関する行が，すべて「多重定義エラー」となっている（エラーの表示形式は，各アセンブラによって異なる）

シンボル「PAR3」の定義を「EQU」ではなく「ASET」に変更してアセンブルしたあとのアセンブルリスト

```
        (4000)      PAR1    EQU     4000H
        (3F00)      PAR2    EQU     3F00H
        (0058)      PAR3    ASET    'X' ·········「PAR3」='X'=0058Hとして定義される
                    ;
0000                        ORG     PAR1 - PAR2
                    ;
0100    1E58                LD      E,PAR3
0102    0E02                LD      C,2
0104    CD0500              CALL    0005H
                    ;
        (005A)      PAR3    ASET    'Z' ·········この行以降は「PAR3」='Z'=005AH
                    ;                            となる
0107    1E5A                LD      E,PAR3
0109    0E02                LD      C,2
010B    CD0500              CALL    0005H
010E    C9                  RET
                    ;
010F    (0000)              END

Errors              0
     エラーなし
```

このプログラムの実行可能なオブジェクト・プログラムを作成し，それを実行した結果

```
A>8EQU↵ ··············作成された「8EQU.COM」を実行

XZ ························XとZの2文字が表示されてCP/Mに戻っている

A>
```

# データ定義

## * DB (Define data Byte: バイトデータ定義)

DB擬似命令については，2.2章「擬似命令」において具体的な解説をしていますが，アーギュメント・フィールドに書かれた絶対値，シンボル，文字列，およびそれらの演算式などのデータを，そのDB擬似命令の位置するロケーション・カウンタのアドレスを先頭に，1バイトずつ，オブジェクト・プログラム中に取り込みます．

数値は今までのものと同じように内部的には，16ビットで取り扱われますが，その下位の8ビット(1バイト)が対象になります．この場合の上位8ビットは，すべて0でなくてはなりません.* クォート[']で囲んだ文字列の場合に限り「16ビット」には関係なく，1バイト単位のアスキーコードの連続として取り扱われます．

このDB擬似命令の使用実例は，DS擬似命令の後に，示します．

## * DW (Define data Word: ワードデータ定義)

前述のDB擬似命令は，アーギュメント・フィールドのデータを，1バイト単位でオブジェクト・プログラム中に取り込みますが，このDW擬似命令は，2バイト単位(2バイトで1ワードと呼ぶ)で取り込みます．つまり，1バイト単位のDB擬似命令が，2バイト単位になったものが，DW擬似命令であるわけです．

数値は，やはり16ビットで扱われますが，DW擬似命令の場合は，2バイト，つまり16ビットの全部を取り込むために，当然のことながら，DB擬似命令の場合のように，上位8ビットがすべて0というような制限はありません．その他のことは，DB擬似命令の場合と同じですが，2文字を越える文字列はエラーとなり，扱うことができません．いずれの場合も，2バイトのデータが下位バイト，上位バイトの順にオブジェクト・プログラム中に取り込まれることに注意してください．この実行例も後ほど示します．

* マイクロソフト社のM80アセンブラでは，すべて1でもよい．

## ＊DS （Define data Storage area： データ領域定義）

アーギュメント・フィールドに書かれた絶対値，シンボル，およびそれらの演算式などの値が示すバイトの数だけ，このDS擬似命令が位置するロケーションから，オブジェクト・プログラム中にメモリエリアを確保します．つまり，DS擬似命令で指定された値のバイト数だけロケーション・カウンタが進み，その次の命令との間に，何のオブジェクトコードも生成されないメモリ空間が確保されるわけです．

次に，DS擬似命令についての簡単な図解を示します．

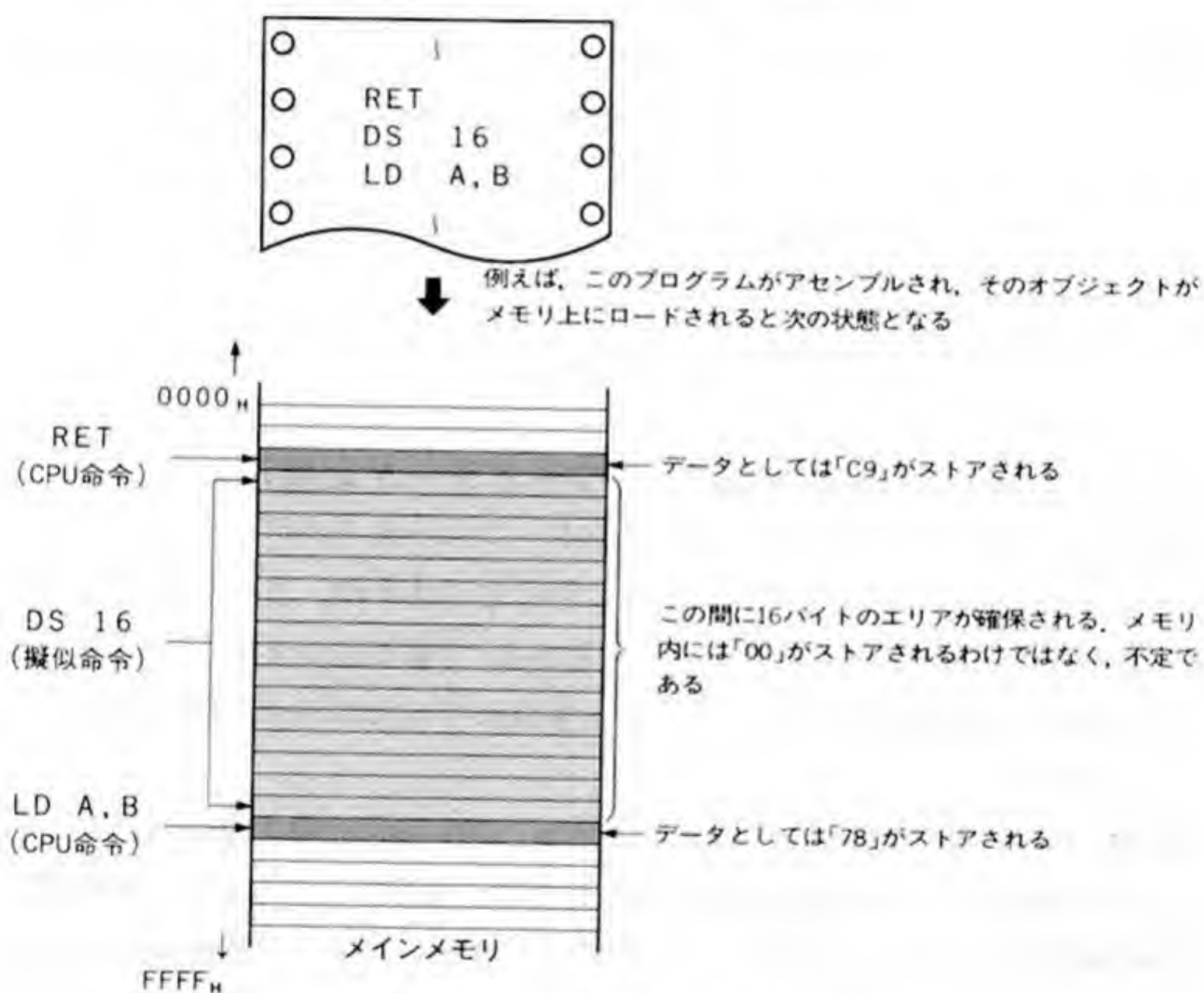



図8-4-3 DS擬似命令の機能

では，DB，DW，DS擬似命令の簡単な使用実例を示しましょう．いろいろな例を示したリストの右側のソース・プログラムと，アセンブルによって生成された左側のオブジェクトコードを，1バイトずつよく対比してみてください．

**図8-4-4　DB, DW, DS擬似命令の使用例**

これはアセンブルリストだが，プログラムとしての意味はない

```
                              ORG     100
                    ;
        (000D)      CR        EQU     0DH
        (000A)      LF        EQU     0AH
        (FA00)      BIOS      EQU     0FA00H
                    ;
0064    0C224142              DB      12,34,'ABCabc123'
        43616263
        313233
006F    0D0A0D0A              DB      0DH,0AH,CR,LF,0AFH AND 0F0H
        A0
0074    32FF                  DB      0032H,0000000011111111B
0076    000000                DB      0F0H + 58H,1234H,BIOS
                              *** value error ***    *      *
0079    0C00                  DW      12,34,'A','BC'
        2200
        4100
        4342
0081    0D00                  DW      0DH,LF,0AFH AND 0F0H
        0A00
        A000
0087    4801                  DW      0F0H + 58H,1234H,BIOS
        3412
        00FA
008D    5634                  DW      123456H
008F    0000                  DW      'abc'
                              *** argument error ***
                    ;
0091                          DS      8
0099    00                    NOP
009A                          DS      LF
00A4    00                    NOP
00A5                          DS      400H
04A5    00                    NOP
                    ;
04A6                          END

Errors              2
```

- CR, LF, BIOS: シンボルの値をこのように定義しておく
- 12,34,'ABCabc123': 000CH 0022H 0041H 0042H……0033H
- 0AFH AND 0F0H: A0H
- それぞれ下位の1バイトのみが有効．上位バイトが00であるのでエラーではない
- ＊これらの値は上位バイトが00でないのでエラー
- 12,34,'A','BC': 000CH 0022H 0041H 4243H……指定が1バイトの場合は，上位バイトに自動的に00がつけられる．オブジェクトには16ビットの下位バイト，上位バイトの順に取り込まれる．(＊文字列も「DB」の場合と異なり，逆になることに注意)
- 123456H……下位の2バイトのみ有効
- 'abc'……文字列が3バイト以上のためのエラー
- DS 8……8バイトのメモリを確保
- DS LF……0AHバイトのメモリを確保
- DS 400H……400Hバイトのメモリを確保
- (0064～008F) このアドレス値は無視してよい
- (0091～04A6) このアドレスは重要　アドレスの進み具合に注目

## 条件アセンブル指定

### ＊IF～ENDIF, IF～ELSE～ENDIF

ソース・プログラム中の，IF および ENDIF 擬似命令の間に書かれている部分のプログラムを，ある条件によってアセンブルしたり，しなかったりすることが可能です．この条件は，IF 擬似命令のアーギュメント・フィールドに書かれた絶対値，シンボル，およびそれらの演算式などの値が，「真」（0以外の値）であるか，「偽」（値が 0）であるかによって，次に示すように判別されます．

〈IF～ENDIF〉

〈IF～ELSE～ENDIF〉

このIF~ENDIF擬似命令の応用には，いろいろなケースが考えられますが，例えば次のような場合に，非常に有効です．

ある同じ働きをするプログラムを，A社のパーソナル・コンピュータのモデルxx用と，B社のyy用に作成する場合を考えましょう．もしこの条件アセンブルの機能を利用しなければ，普通は，xx用と，yy用の2本の独立したソース・プログラムを作成しなければなりません．

ところがこのような場合，両者に違いがある部分を，IF~ENDIF擬似命令で囲めば，1本のソース・プログラムにまとめることが可能です．そしてアセンブル時に，ソース・プログラムの冒頭で，xxか，yyかを指定することにより，任意の側のオブジェクト・プログラムを得ることができます．

では，IF~ENDIF擬似命令の使用例を示しましょう．この例題プログラムは「X」の1文字を表示するだけの極めて簡単な内容です．ただしこのプログラムは，IF~ENDIF擬似命令を利用して，CP/Mまたは$N_{88}$-BASICのいずれで実行させるか，およびZ-80または8080のいずれのアセンブラを使ってアセンブルするかを，条件選択できるようになっています．

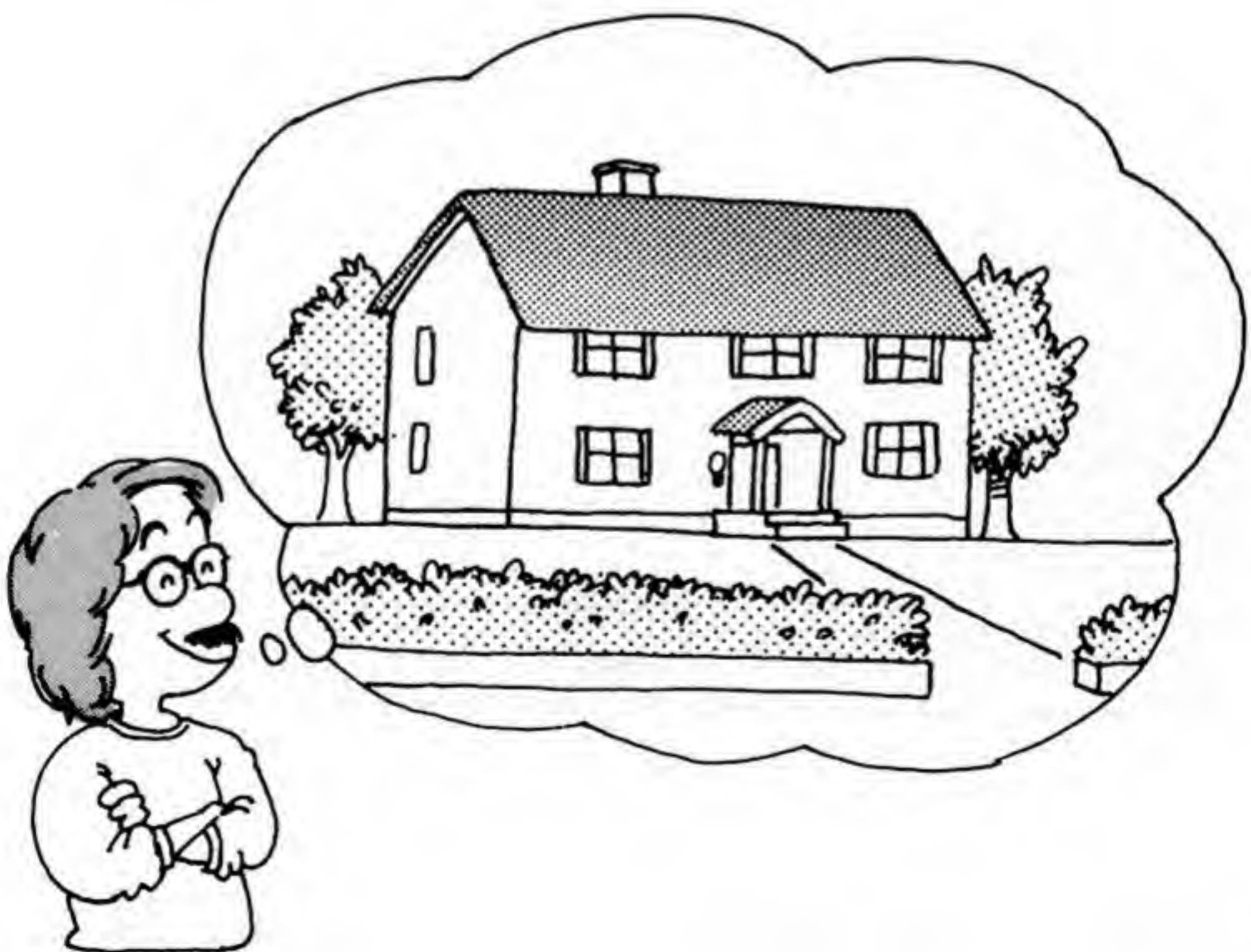
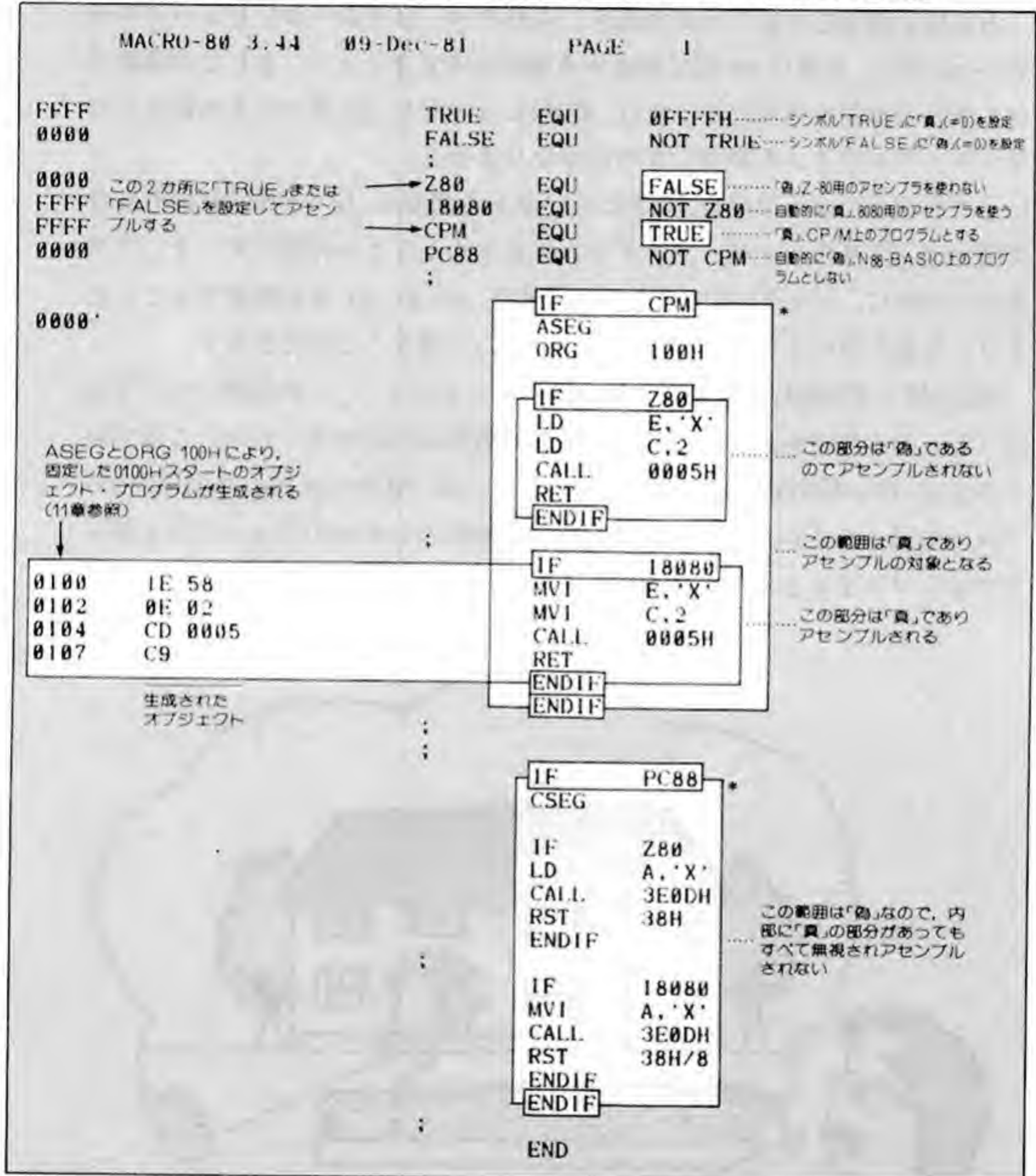
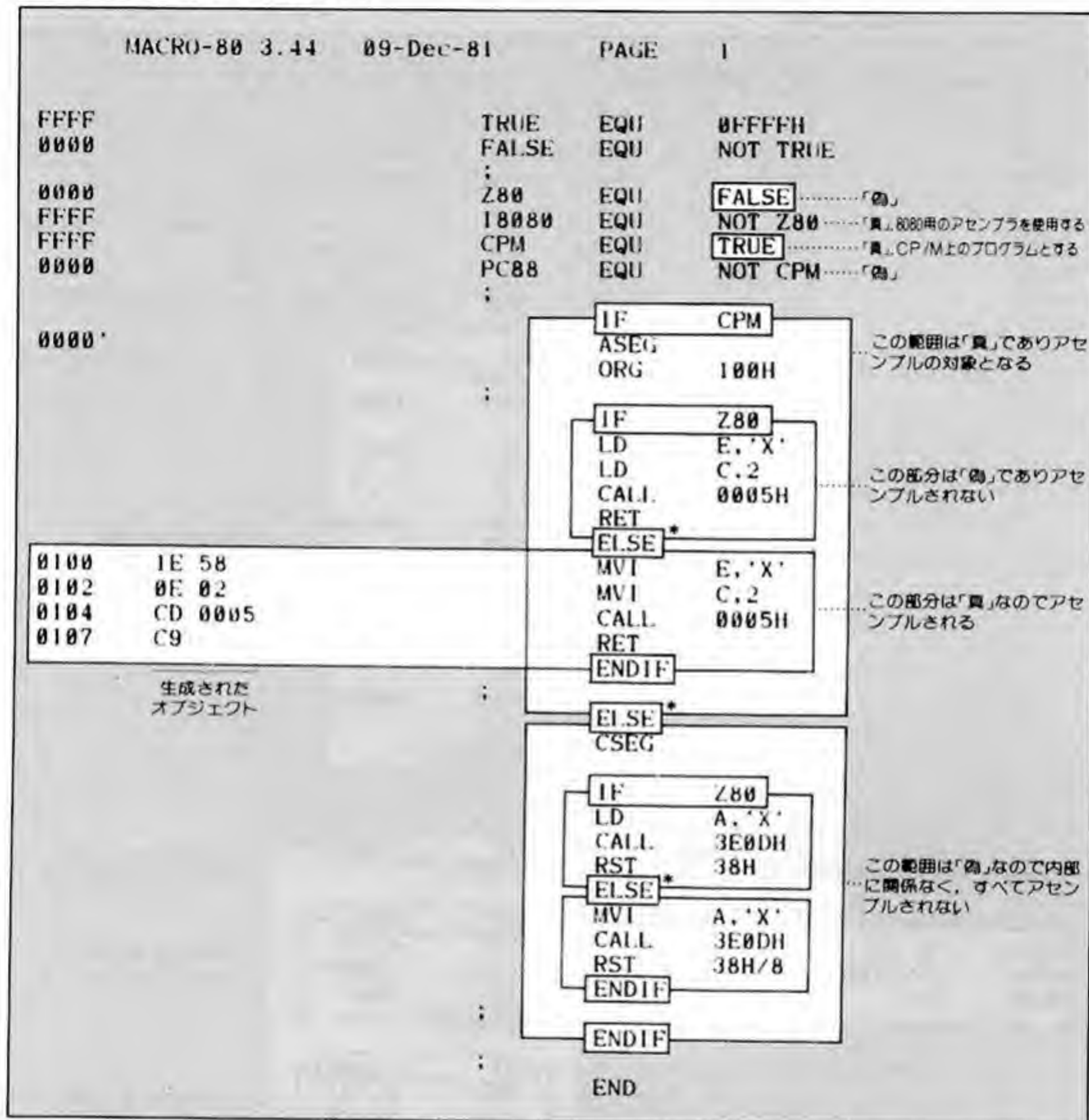



図8-4-5　IF〜ENDIFの使用例

CP/M上で実行するプログラムで，8080用アセンブラを使ってアセンブルするように条件を設定したもの．アセンブル後のアセンブルリストで示すので，ソース・プログラム冒頭の条件設定と，生成されているオブジェクト・プログラムに注目

＊CP/Mのアセンブラ「ASM」は，このプログラムのようにIF−ENDIFの入れ子(ネスティング)をしたものはアセンブルできない．「ASM」を使用する場合は，ネスティングせずにそれぞれが単独になるように変更するとよい．

N88-BASIC上で実行するプログラムで，Z-80用のアセンブラを使ってアセンブルするように条件を設定したもの

IF~ELSE~ENDIFを使った例．CP/M上で実行するプログラムで，8080用のアセンブラを使ってアセンブルするように条件を設定したもの

# ファイル関係指定

## ＊ END （プログラムの終了指定）

END擬似命令は，ソース・プログラムの終了を指定します．つまり，ソー

＊CP／Mのアセンブラ「ASM」には～ELSE～の機能はない

ス・プログラム中で，END 擬似命令が表れた時点を，ソース・プログラムの終了とみなします．よって，それ以降に何が書かれていようと，アセンブラはすべて無視します．

この擬似命令の使い方には次に示す2通りがあります．

```
(a) END
(b) END            0100H — 絶対値の0100Hは一例であり，シンボル
                         や演算式などでもよい
```

(a)と(b)の違いは，CP/Mアセンブラ「ASM」などのインテルHEX形式のオブジェクト・プログラムを生成するものでは，そのオブジェクト・プログラムの最終ブロック(プログラム本体には関係しない部分)に表れます．

アセンブラによっては，このEND擬似命令を書かなくてもよいものもありますが，ソース・プログラムのリストの終りを，読む人に明確に示す意味で，(a)の形式の「END」だけは書いておいた方がよいでしょう．

オブジェクト・プログラム自身の内部的なスタート・アドレス(ロード・アドレスの先頭)は，ソース・プログラムのORG擬似命令やリンクロード時(リロケータブル・アセンブラの場合)に指定されるもので，これに基づいてオブジェクト・プログラムが指定されたアドレス順に生成されます．

(b)の形式での，アーギュメント・フィールドに書かれる値(上の例では$0100_H$)も「スタート・アドレス」と呼ばれます．しかしこの場合の「スタート・アドレス」は，(a)のように生成されるオブジェクト・プログラムには直接関係なく，あくまで外部的なものであり，別の目的に利用するためのものですので，混同しないように注意してください．*

では，END擬似命令のいくつかの使用実例を示しましょう．まず，END擬似命令以降の記述は，すべて無視されることの例です．

* 例えば，「M80」アセンブラでは，(b)の形式で「スタート・アドレス」を指定しておくと，「L80」によるリンク作業の後で，メモリ上に生成された実行可能なオブジェクト・プログラムを自動的に実行することが可能になる．このほか，アセンブラやローダによっては，(b)の形式のEND命令はいろいろなことに利用することができるのでそれぞれのアセンブラのマニュアルを参照するとよい．

2~5章で多用したメッセージ出力のプログラム

**図8-4-6　END擬似命令の後が無視される例**

```
BDOS    EQU     0005H
CR      EQU     0DH
LF      EQU     0AH
EOS     EQU     00

        ORG     100H
START:
        LD      HL,MESG
LOOP:
        LD      A,(HL)
        OR      A
        RET     Z
        PUSH    HL
        CALL    CHROUT
        POP     HL
        INC     HL
        JP      LOOP

        END  ……………途中に「END」を書いた
CHROUT:
        LD      C,2
        LD      E,A
        CALL    BDOS
        RET


MESG:   DB      CR,LF,'Good Morning',CR,LF,EOS

        END
```

そのソース・プログラムをアセンブルした結果のアセンブルリスト

```
        (0005)      BDOS    EQU     0005H
        (000D)      CR      EQU     0DH
        (000A)      LF      EQU     0AH
        (0000)      EOS     EQU     00

                            ORG     100H

                    START:
0100    210000              LD      HL,MESG ………*
                            *** undefined symbol ***

                    LOOP:
0103    7E                  LD      A,(HL)
0104    B7                  OR      A
0105    C8                  RET     Z
0106    E5                  PUSH    HL
0107    CD0000              CALL    CHROUT ………*
                            *** undefined symbol ***

010A    E1                  POP     HL
010B    23                  INC     HL
010C    C30301              JP      LOOP

010F    (0000)              END

Errors          2
```

*ENDの行以降に出てくるラベルなので，「定義されていないラベル(シンボル)を使用した」とエラー表示される

これ以降がすべて無視され，アセンブルされていない．アセンブルがここで終わっている

次に，CP/Mアセンブラ「ASM」について，(a)および(b)の形式で記述した場合の，それぞれのオブジェクト・プログラムの違いを，インテルHEX形式のオブジェクトファイルをタイプアウトして示しましょう.*

ここでの例題プログラムとしては，2，3，4，5章でおなじみの，メッセージ出力のプログラムを使っています.

**図8-4-7 「END」のみ記述した場合のオブジェクトファイル**

2～5章で多用したメッセージ出力のプログラム

```
BDOS     EQU     0005H
CR       EQU     0DH
LF       EQU     0AH
EOS      EQU     00

         ORG     100H
START:
         LD      HL,MESG
LOOP:
         LD      A,(HL)
         OR      A
         RET     Z
         PUSH    HL
         CALL    CHROUT
         POP     HL
         INC     HL
         JP      LOOP

CHROUT:
         LD      C,2
         LD      E,A
         CALL    BDOS
         RET

MESG:    DB      CR,LF,'Good Morning',CR,LF,EOS

         END
```

このソース・プログラムを「8END」としてアセンブルし，HEX形式のオブジェクト・プログラムを生成する

```
A>TYPE 8END.HEX ↵ ……………そのHEX形式のオブジェクト・プログラムをタイプアウトする

:1C0100002116017EB7C8E5CD0F01E123C303010E025FCD0500C90D0A476F6F6477
:0B011C00204D6F726E696E670D0A00C7
:0000000000 ←── この行のロード・アドレスに注目.「0000」となっている
     ↑                                         ↑
 ロード・アドレス                      オブジェクト・プログラム部
A>
```

*インテルHEX形式については，APPENDIX 2を参照.

**図8-4-8　END擬似命令にスタート・アドレスをつけて記述した例**

先のソース・プログラムのEND擬似命令に「スタート・アドレス」を設定する

このソース・プログラムを「8ENDST」としてアセンブルし，HEX形式のオブジェクト・プログラムを生成する

```
A>TYPE 8ENDST.HEX ↵
:1C0100002116017EB7C8E5CD0F01E123C303010E025FCD0500C90D0A476F6F6477
:0B011C00204D6F726E696E670D0A00C7
:00010000FF
A>
```

そのタイプアウト

ロード・アドレス

この行のロード・アドレスに注目．END擬似命令で指定した0100Hになっている．しかしオブジェクト・プログラム部には何の影響もない

# 9 実用プログラムの作成

本章では，今までの知識を総合して，何か適当な実用プログラムを作成してみましょう．「実用プログラム」といっても，あまり大きなものでは全体を見通すことが困難になり本書の目的から外れますので，ここではメモリダンプ・プログラムの簡易版を選んでみました．

ここではひとつのソース・プログラムから，CP／M上で実行できるものと，PC-8801 の $N_{88}$-BASIC 上で実行できるものとのどちらのオブジェクト・プログラムも作成可能なようにします．つまり 8 章で解説した擬似命令，IF～ENDIF の機能を利用してプログラミングします．

また，CP／Mのアセンブラ「ASM」でも実習が可能なように，ソース・プログラムは Z-80 のニーモニックで書いたものだけでなく，インテル形式の 8080 ニーモニックで書いたものも用意しました．Z-80 は，8080 の上位コンパチブルですので，8080 アセンブラで作られたオブジェクト・プログラムをそのまま実行できることは，ご存知でしょう．

なお，ここで作成するプログラムは，次の 10 章でのデバッグ作業において，デバッグ対象のプログラムとして取り上げ，主要ルーチンの動きなどをテストしますので参照してください．

# 9/1 メモリダンプ・プログラム簡易版

メモリ内容をダンプするプログラムの代表的な表示形式は，CP/Mに標準装備のデバッガ，「DDT」のD(Dump)コマンドでしょう．この表示形式は，本書のリストの随所で見られますが，次のようなものです．

図9-1-1　DDTのDumpコマンドによる表示

```
                          .
                          .
                          .
0160 01 0E 02 5F CD 05 00 C9 0E 01 CD 05 00 C9 0D 0A ... ............
0170 31 2E 20 4D 6F 72 6E 69 6E 67 0D 0A 32 2E 20 4E 1. Morning..2. N
                          .
```

アドレス表示部　メモリ内容表示部(16進)　16進表示に対応するアスキー表示部

このように，16バイト分のメモリ内容が1行の中央に16進で表示され，左端には，その行の最初のデータのメモリ・アドレスが表示されています．また，16進表示のメモリデータの右側には，それぞれのバイトのデータをアスキーコードとして見た場合の，それに対応した文字が示されています．例えば，16進表示が「4D」のデータは，アスキー表示では「M」，同じく「6F」は小文字の「o」というように表示されます．また，アスキーコードとして見た場合に，文字として表示できないものに対しては，[.]が表示されます．

このDDTのようなダンプ・プログラムを作るのは，さほど難しくはないのですが，ここでは，プログラムが長くならないように，できるだけコンパクトにまとめたものを作成します．よって，1行に16バイトを表示したり，アスキー表示をしたりすることは意識的に避けています．ここでのプログラムに，それらの機能を付加することを，後ほど各自で試みるのもよいでしょう．

# 作成するダンプ・プログラムの仕様

では，次のようなコンパクトなダンプ・プログラムを作ってみましょう．

| | |
|---|---|
| プログラムを起動するとまずプロンプト[－]が表示される | － |
| 次に，内容を見ようとする(ダンプする)メモリのアドレスを入力してリターンする．ただし，4桁入力した場合はリターンの必要はない | －100↵ |
| アドレス値とその1バイトのメモリ内容が表示される．その後リターンを入力すると， | －100<br>0100：　C3↵ |
| 次のアドレスのメモリ内容が同様に表示される．このようにリターンの入力を続けると，次々とダンプされていく | －100<br>0100：　C3<br>0101：　03↵ |
| リターンのかわりに，その他のキーを入力すると，再び最初のプロンプト表示に戻るので，新しいダンプアドレスの入力ができる | ︙<br>0102：　DA↵<br>0103：　81↵<br>－ |
| ダンプアドレスの入力に，0－F以外の文字を入力すると[？]が表示され，再入力となる | ︙<br>－40G↵<br>?<br>－ |

**図9-1-2　作成するダンプ・プログラムの仕様**

このダンプ・プログラムは，ダンプするアドレスの範囲を，from-to(開始アドレス-終了アドレス)で指定する機能はありません．開始アドレスだけが指定可能であり，それに続くアドレスは，リターンキーを押すことにより1バイトずつ次々とダンプしていくことにします．このときに，[E]あるいは[e]を入力すると，ダンプ・プログラムが終了します．その他のキーの入力があった場合は，再度ダンプ開始アドレスの入力待ち状態となります．

# 9/2 全体の構成

このダンプ・プログラムを実現するには，いろいろなプログラミングの方法があると思いますが，ここでは，全体を次のように構成してみました．メインルーチンの骨子をフローチャートの形式で次に示しましょう．

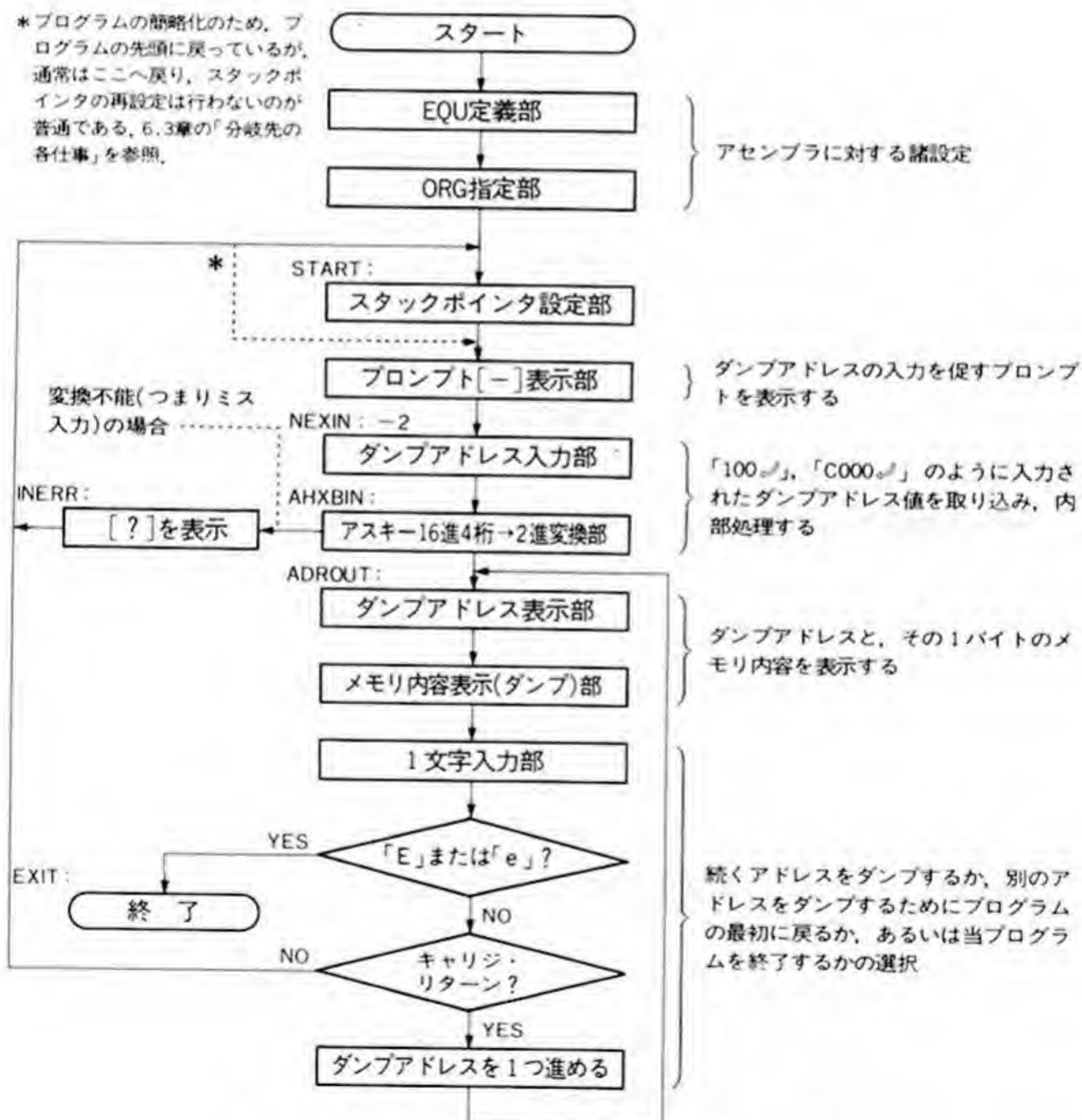



図9-2-1　作成するダンプ・プログラムの構成

構成自体については，特に複雑な箇所もありませんので，前のページの図からすぐに理解できることと思います．構成はこれでよいとしても，これをプログラミングするには，次に示す主要な2つの処理ルーチンが必要です．

**(a) キー入力されたダンプしようとするアドレス(メモリ上にアスキーコードで取り込まれている)を2バイトのバイナリ形式に変換するルーチン**

例えば，ダンプ・アドレスを"C08F"とキー入力した場合，そのデータは，メモリ上のアドレス入力バッファ(プログラムによって5バイト用意されている)に，アスキーコードで，$43_H$，$30_H$，$38_H$，$46_H$ という形で取り込まれています．これを，CPUでアドレスとして処理するには，「$C08F_H$」という2バイトのバイナリ形式に変換しなければなりません．この処理を「アスキー16進4桁 ⇒ 2進2バイト変換」と呼ぶことにしましょう．

**(b) メモリ上，あるいはレジスタ上のバイナリ形式の1バイト単位のデータを，その16進「読み」どおりのアスキーコードに変換するルーチン**

例えば，メモリ上，あるいはレジスタ上に，$41_H$という1バイトのデータがある場合，これをCRTディスプレイに"41"と表示しなければなりません．このためには，プログラム内に16進の「読み」どおりのアスキーコードに変換するルーチンが必要です．

各章の例題でおなじみの，1文字出力ルーチン(CHROUTなどのラベルがつけられている)に，このデータ$41_H$を入力しても，"41"とは表示されず，"A"と表示されてしまいます．これは，1文字出力ルーチンが，そのルーチンへの入力データをアスキーコードとみて，それに対応するアスキー文字を表示するからです．これは，アスキーコードの$41_H$が，文字"A"を表しているからです．しかし，ここでは，データ$35_H$(アスキーコードで$35_H$は「5」を表す)であれば，"5"ではなく，"35"と表示したいわけです．この処理を，「2進1バイト ⇒ アスキー16進変換出力」と呼びましょう．

この2つの処理を行うルーチンが，このプログラムで最も手のかかるところであり，その他の部分は，今までの章の例題で実習したことなどを応用すれば，比較的簡単にできると思います．

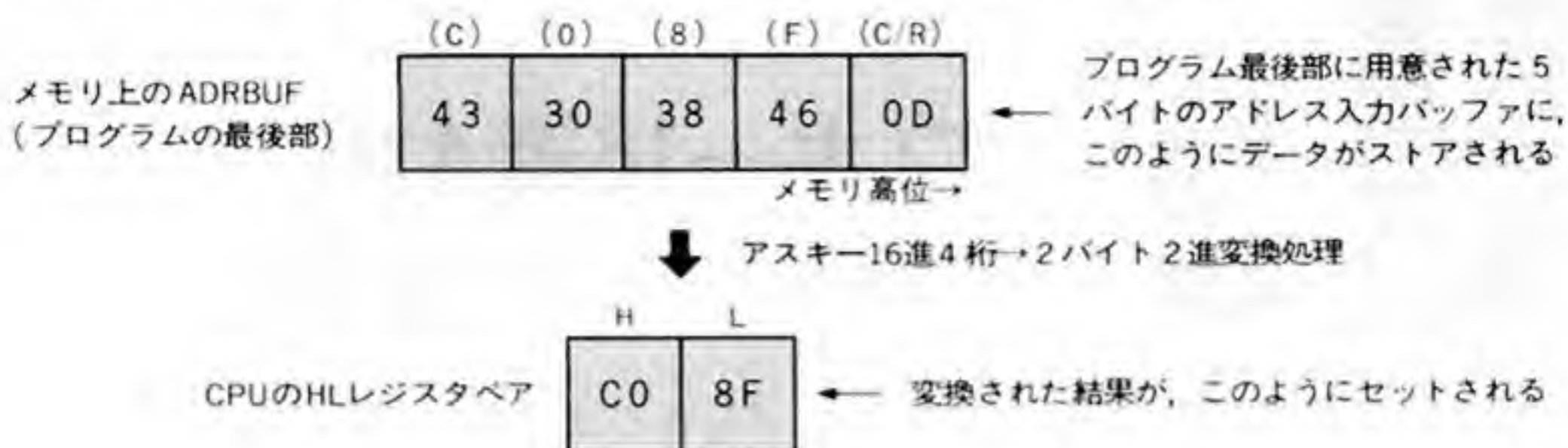
# 9/3 アスキー16進4桁→2進2バイト変換

このダンプ・プログラムでは、ダンプしようとするアドレスは、例えば、

| | | | | |
|---|---|---|---|---|
| $C08F_H$ | 番地であれば, | C08F | | |
| $0100_H$ | 番地であれば, | 100 ↵ | あるいは | 0100 |
| $00E5_H$ | 番地であれば, | E5 ↵ | あるいは | 00E5 |

のようにキー入力します．1文字キー入力ルーチン(ラベル「CHRIN：」)は，入力された文字のアスキーコードを発生しますので，入力されたアドレス値は，プログラムの最後部に設けたラベル「ADRBUF：」のアドレス入力バッファに，アスキーコードで格納されます．格納されたデータの最後には，必ずキャリジ・リターンのコード「$0D_H$」が格納されることにも注目してください．これらの処理は，**図9-1-3**「ダンプ・プログラムの構成図」の「ダンプ・アドレス入力部」で行っています．

このようにして，アスキーコードでアドレス入力バッファに入力されたダンプ・アドレスは，アスキー16進4桁 ⇒ 2進2バイト変換の処理(ラベル「AHXBIN：」から始まるルーチン)により，CPUのHLレジスタペアにバイナリ形式でセットされます．この処理により，次のステップの仕事である，ダンプしようとするアドレスのメモリ内容を取り出すことができるようになるわけです．この部分の概略を次の図で示しましょう．

ダンプするアドレスとして「C08F」が入力された場合(つまりC08F$_H$番地)

ダンプするアドレスとして「5↵」が入力された場合(つまり0005番地)

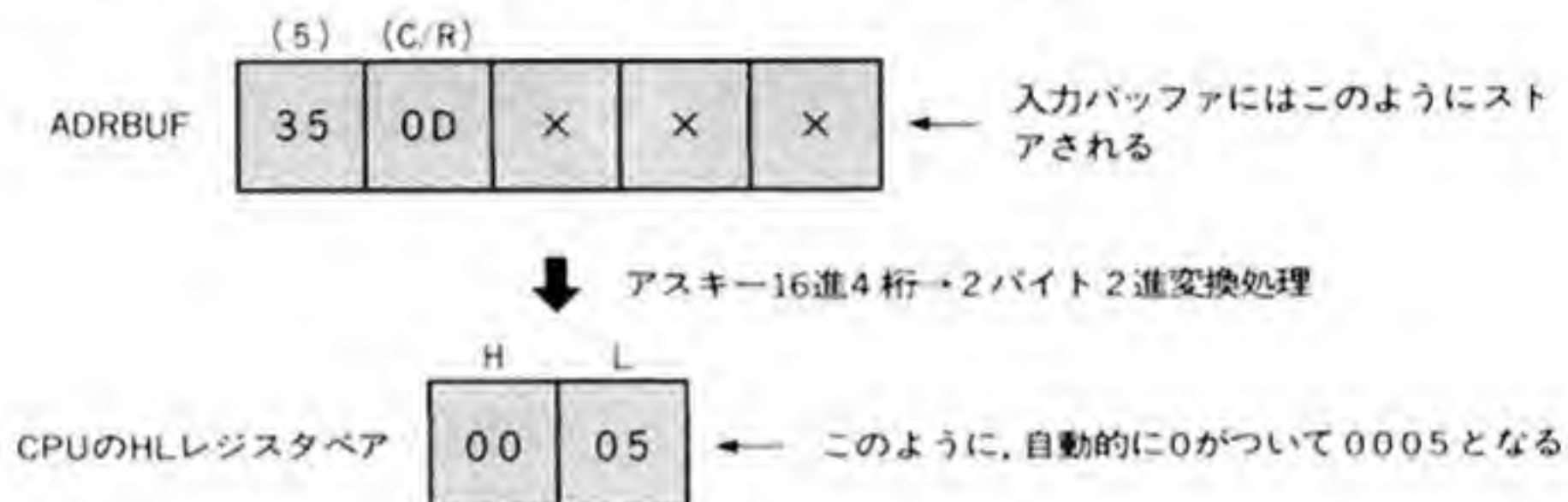



**図9-3-1　アスキー16進4桁→2進2バイト変換の概略**

実際のルーチンは，ソース・プログラム(**図9-6-2**または**図9-6-9**のアセンブルリスト)を参照してください．このルーチンについては，10章(**図10-3-5**)において，デバッガによって変換処理ルーチンの全ステップのCPUの動きを追っていますので，そちらも参照してください．

# 9-4 2進1バイト→アスキー16進変換出力

メモリや,CPUのレジスタの1バイトのデータを,アスキー16進形式で表示するためには,前節と逆の,2進1バイト ⇒ アスキー16進変換を行う必要があります.

メモリや,レジスタのデータは,2進,つまりバイナリ形式で格納されています.例えば,ある1バイトのデータが「$C0_H$」の場合,これをCRTディスプレイなどに表示するには,〝C〟(アスキーコード$43_H$)の文字と,〝0〟(アスキーコード$30_H$)の文字とに変換し,この2文字を連続して〝C〟〝0〟と出力することにより,〝C0〟と表示できるわけです.この部分の概略を次の図で示しましょう.

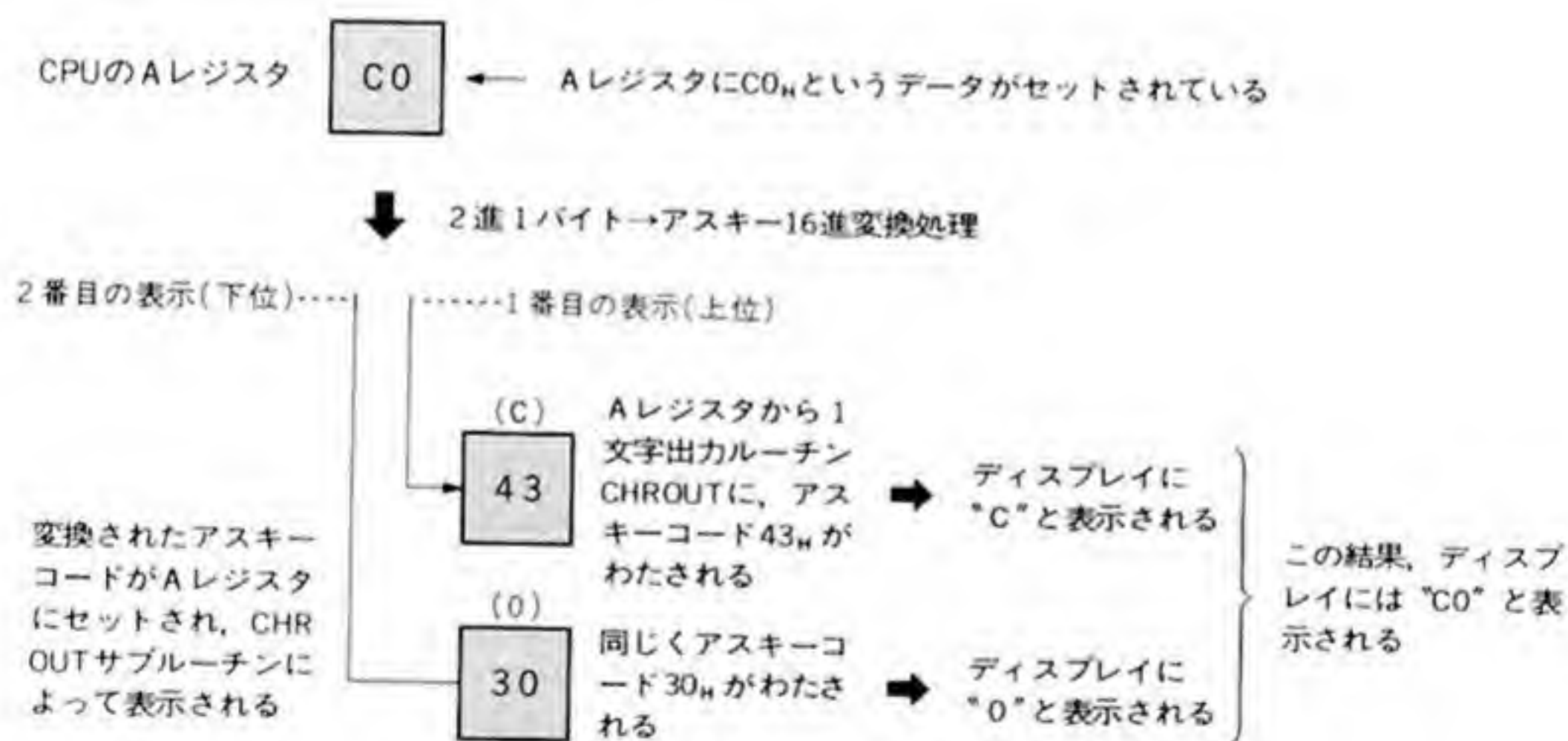



**図9-4-1 2進1バイト→アスキー16進変換出力の概略**

実際のルーチンは,ソース・プログラム(**図9-6-2**または**図9-6-9**のアセンブルリスト)を参照してください.なおこのルーチンについても,10章(**図10-2-3**)において,その処理が行われるまでの全ステップのCPUの動きを,デバッガで追っていますので,そちらも参照してください.

# 9/5 ソース・プログラムの構成

メモリダンプ・プログラムのソース・プログラムの構成を次の図で示します．

| メインルーチン<br>（図9-1-3の構成図を参照） |
| --- |
| 各サブルーチン<br><br>HEXOUT：　2進1バイト→16進変換出力サブルーチン<br>CHROUT：　1文字出力サブルーチン<br>CRLFOUT：復帰／改行出力サブルーチン<br>CHRIN：　1文字キー入力サブルーチン |
| バッファ／スタックエリア<br><br>ADRBUF（5バイト）アドレス入力バッファ<br>- - - - - - - - - -<br>16レベルのスタックエリア<br><br>STACK（32バイト） |

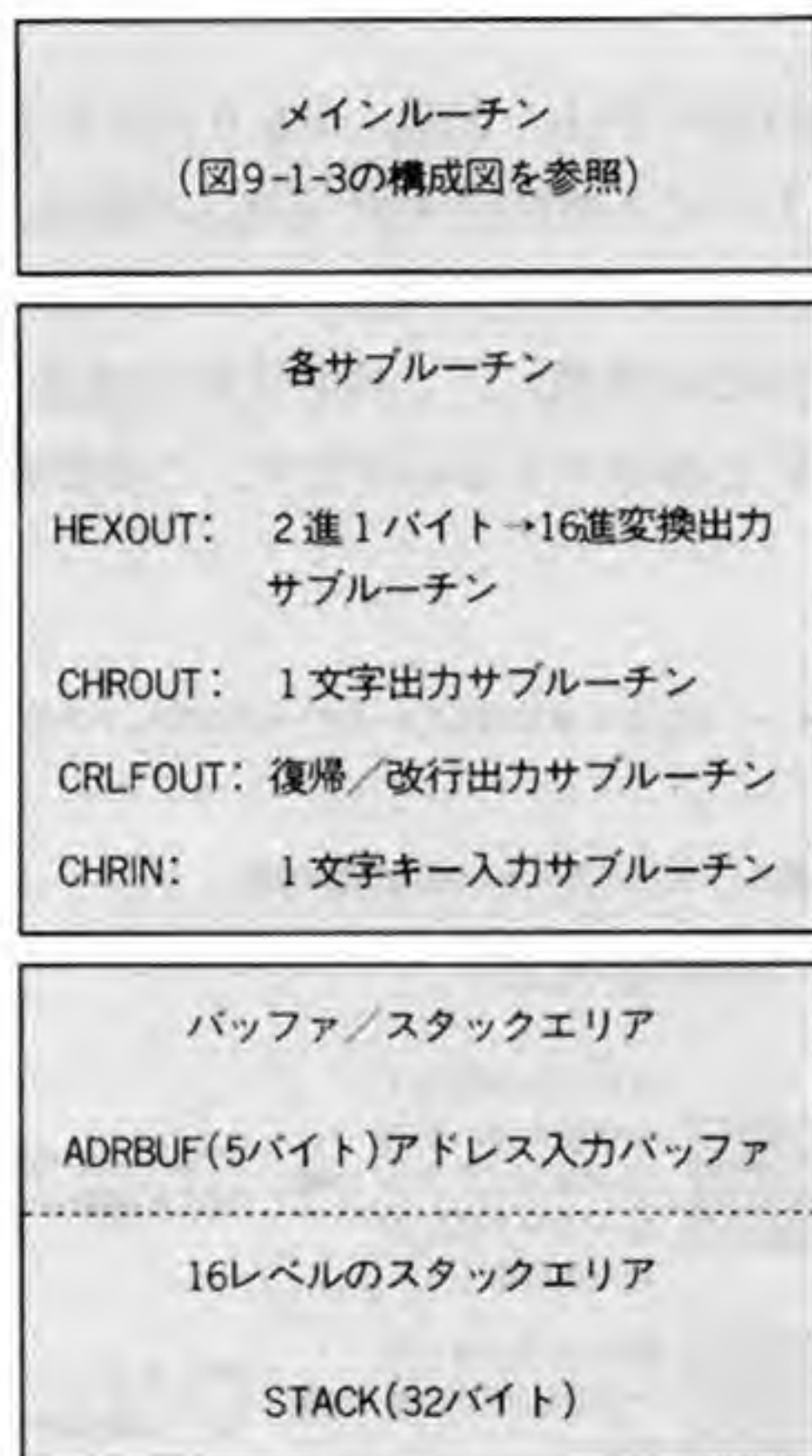



**図9-5-1　ソース・プログラムの構成の概要**

ソース・プログラムは，**図9－2－1**に示した流れがメインルーチンとなり，それらが動作するために必要なサブルーチンがその後に続きます．

**図9－2－1**および**図9－5－1**では，ソース・プログラム上で，その部分に相当するラベルが存在するものについては，そのラベル名を書き加えてあり

ますので，ソース・プログラムと対比してみてください．なお，**図 9－2－1**でダンプ・アドレス入力部の肩に「NEXIN：－2」と書かれているのは，ラベル「NEXIN：」の2バイト前，つまり1ステップ前のことです．

メインルーチンで呼び出されるサブルーチンには，本章で新たに作成した2進1バイト ⇨ アスキー16進変換出力サブルーチン(ラベルは，HEXOUT：)と，今までの各例題でおなじみの1文字出力サブルーチン(ラベルは，CHROUT：)，復帰／改行出力サブルーチン(ラベルは，CRLFOUT：)，それに1文字キー入力サブルーチン(ラベルは，CHRIN：)があります．

プログラムの最後部には，5バイトのアドレス入力バッファと，16レベル，つまり16段階のネスティングが可能なスタックエリア(32バイト)が設けてあります．全体のソース・プログラムは，次節の**図 9－6－2**，および**図 9－6－9**で示すアセンブルリストを参照してください．

# 9/6 アセンブルおよび実動テスト

まずエディタにより，メモリダンプ・プログラムのソース・プログラムを作成します．このソース・プログラムは，本章の冒頭でも触れたようにCP/M上でもPC-8801上でも動作するようにIF～ENDIF擬似命令を使ったものとしました．

そこでCP/Mで実行するのかBASICで実行するのかにより，ソース・プログラムの冒頭にあるIF宣言部「CPM」の真偽をTRUE(真)，あるいはFALSE(偽)に定義した後，アセンブルします．

アセンブル・エラーが出なければ，ローダにより，実行可能なオブジェクト・プログラムや，インテルHEX形式のオブジェクト・プログラムを生成する作業を行います．なお，本節ではZ-80のニーモニックで書いたプログラム(M80による)を紹介した後，インテル形式の8080ニーモニックで書いたプログラムも紹介します．

## CP/M上で実行するオブジェクト・プログラムをM80，L80で作成

ではまず，M80，L80により，CP/M上で実行するためのオブジェクト・プログラムを生成してみましょう．

M80アセンブラを実行する前に，IF宣言部の「CPM」を「TRUE」にします．また，M80がリロケータブル・アセンブラのため，ORG擬似命令の前にASEG擬似命令を挿入しておきます．ASEG擬似命令は，11章「リロケータブル・マクロアセンブラの概念と使い方」で解説しますが，「アブソリュート・セグメント」の意味であり，リロケータブルではなく，絶対アドレスを持ったオブジェクト・プログラムを生成することをM80に指示します．*

*ASEGを使わずにリロケータブルのままにしておき，リンクローダの実行の際に絶対アドレスを指定する方法もある．

このソース・プログラムのファイル名を「DUMPZ.MAC」として，M80，L80によるアセンブルおよびロードの実行例を示します．

図9-6-1　M80，L80によるCP／M用オブジェクトの生成

```
A>DIR DUMPZ.* ↲ ………実行前のすべての「DUMPZ」ファイルの確認

A: DUMPZ    MAC
   ソース・プログラムのみ存在

A>M80 DUMPZ,DUMPZ=DUMPZ/Z ↲ ……アセンブラM80の実行．最後「/Z」は，Z-80のニーモニックを
                                   アセンブルするためのスイッチ
No Fatal error(s)
アセンブル終了．アセンブル・エラーはなし
A>DIR DUMPZ.* ↲ ………アセンブル後の「DUMPZ」ファイルの確認

A: DUMPZ    MAC : DUMPZ    PRN : DUMPZ    REL
                  生成されたアセンブル  生成されたリロケータ
                  リスト               ブル・オブジェクト・プログラム
A>L80 DUMPZ,DUMPZ/N/E ↲ ……リンクローダL80を実行して，「DUMPZ.REL」から実行可能なオブジェクト
                            を生成する
Link-80  3.44  09-Dec-81  Copyright (c) 1981 Microsoft

Data  ┌→0100    01E3←┐  <  227>←プログラム全体のバイト数
  プログラムのスタート・アドレス
40791 Bytes Free       エンド・アドレス
[0100    01E3      1]
リンクロード終了
A>DIR DUMPZ.* ↲ ………「DUMPZ」ファイルの確認

A: DUMPZ    MAC : DUMPZ    PRN : DUMPZ    REL : DUMPZ    COM
A>                                               実行可能なオブジェクト・プロ
                                                 グラムが生成されている
```

この実行例では，M80によるアセンブルで，アセンブルリスト(.PRN)と，オブジェクト・プログラム(.REL)がディスク上に生成されました．さらにL80リンクローダにより，オブジェクト・プログラム「DUMPZ.REL」から，実行可能なオブジェクト・プログラム「DUMPZ.COM」が生成されています．つまり，

|  | ( L80 ) |  |
|---|---|---|
| .REL | ⇨ | .COM |
|  |  | (実行可能な純マシン語) |

の変換が行われました．

では，生成されているアセンブルリストをタイプアウトしてみましょう．条件アセンブルにより，アセンブルされている箇所と，されていない箇所に注目してください．

図9-6-2　生成されたアセンブルリスト．CP／M用オブジェクトの場合

```
        MACRO-80 3.44   09-Dec-81       PAGE    1

                                ;---------------------------------------;
                                ;         MEMORY DUMP PROGRAM           ;
                                ;    for [hajimete-yomu-assembler]      ;
                                ;---------------------------------------;
                                ;
FFFF                            TRUE    EQU     0FFFFH
0000                            FALSE   EQU     NOT TRUE
                                ;
FFFF                            CPM     EQU     TRUE      …… 「真」．CP/M上のプログラムを作る
0000                            PC88    EQU     NOT CPM   …… 自動的に「偽」となる
                                ;
0005                            BDOS    EQU     0005H     …… CP/Mシステムコールの
3583                            PCIN    EQU     3583H        エントリー・ポイント
3E0D                            PCOUT   EQU     3E0DH
000D                            CR      EQU     0DH       …… 復帰コード
000A                            LF      EQU     0AH       …… 改行コード
                                ;
0000'                                   ASEG   …… リロケータブルではなく，固定アドレスを持った
                                                  オブジェクトを生成するための指定
                                        IF      CPM
                                        ORG     100H      …… CP/Mの場合は0100Hスタート
                                        ENDIF
                                ;
                                        IF      PC88
                                        ORG     9000H     …… アセンブルされない
                                        ENDIF
                                ;---------------------------------------
                                ;          main routine
                                ;---------------------------------------
0100                            START:
                                        IF      CPM
0100    31 01E3                         LD      SP.STACK  …… スタック・ポインタの設定
                                        ENDIF
                                ;
0103    11 01BE                         LD      DE.ADRBUF …… DEレジスタペアに，アドレスバッ
                                                             ファの先頭アドレスをセット
0106    CD 01A7                         CALL    CRLFOUT   …… 復帰/改行を出力
0109    3E 2D                           LD      A.'-'     | ダンプ・アドレスの入力を促す
010B    CD 019C                         CALL    CHROUT    | プロンプト「－」を出力
                                ;
010E    06 04                           LD      B.4       …… ダンプ・アドレスの入力文字数を4桁に規制
                                                             するカウンタ
0110                            NEXIN:
0110    CD 01B2                         CALL    CHRIN     |
0113    12                              LD      (DE).A    | キー入力されたダンプ・アドレスをアド
0114    FE 0D                           CP      CR        | レス・バッファに格納していく
0116    CA 0121                         JP      Z.AHXBIN  | キャリジ・リターンが入力されると，ア
0119    13                              INC     DE        | スキー16進→2進2バイト変換ルーチンへ
011A    05                              DEC     B         | ジャンプする．4桁入力されると次のステ
011B    C2 0110                         JP      NZ.NEXIN  | ップへ
011E    3E 0D                           LD      A.CR      | 4桁入力されるとその後に自動的にキャ
0120    12                              LD      (DE).A    | リジ・リターン(0DH)が書き込まれる
```

```
                                ;
                                ;------ this routine converts ASCII 2byte
                                ;       digits into BINARY HEX.
                                ;   DE=ASCII data pointer  HL=converted data
0121                            AHXBIN:
0121    21 0000                         LD      HL,0
0124    11 01BE                         LD      DE,ADRBUF
0127                            AHXBI1:
0127    1A                              LD      A,(DE)
0128    FE 0D                           CP      CR
012A    CA 0152                         JP      Z,ADROUT
012D    29                              ADD     HL,HL
012E    29                              ADD     HL,HL
012F    29                              ADD     HL,HL
0130    29                              ADD     HL,HL
0131    CD 013D                         CALL    HCONV
0134    D2 0147                         JP      NC,INERR
0137    85                              ADD     A,L
0138    6F                              LD      L,A
0139    13                              INC     DE
013A    C3 0127                         JP      AHXBI1
013D                            HCONV:
013D    D6 30                           SUB     30H
013F    FE 0A                           CP      0AH
0141    D8                              RET     C
0142    D6 07                           SUB     7
0144    FE 10                           CP      10H
0146    C9                              RET
                                ;
                                ;------ input data is not valid hex value
0147                            INERR:
0147    CD 01A7                         CALL    CRLFOUT
014A    3E 3F                           LD      A,'?'
014C    CD 019C                         CALL    CHROUT
014F    C3 0100                         JP      START
                                ;
                                ;------ address out
0152                            ADROUT:
0152    CD 01A7                         CALL    CRLFOUT
0155    7C                              LD      A,H
0156    CD 0184                         CALL    HEXOUT
0159    7D                              LD      A,L
015A    CD 0184                         CALL    HEXOUT
015D    3E 3A                           LD      A,':'
015F    CD 019C                 *       CALL    CHROUT
0162    3E 20                           LD      A,' '
0164    CD 019C                         CALL    CHROUT
                                ;
                                ;------ memory data out
0167    7E                              LD      A,(HL)
0168    CD 0184                         CALL    HEXOUT
                                ;
                                ;------ check continue or new address
016B    CD 01B2                         CALL    CHRIN
016E    FE 45                           CP      'E'
0170    CA 0181                         JP      Z,EXIT
0173    FE 65                           CP      'e'
0175    CA 0181                         JP      Z,EXIT
0178    FE 0D                           CP      CR
017A    C2 0100                         JP      NZ,START
```

AHXBIN〜HCONV RET：アスキー16進4桁・2進2バイト変換ルーチン
プログラム最後部のアドレスバッファ「ADRBUF」にアスキーコードで格納されているダンプ・アドレス値を，2進2バイトに変換して，HLレジスタペアにセットする機能を持つ．アドレスバッファには，1〜4桁のアドレス値が格納されており，それぞれの最後にはキャリッジ・リターンのコード（0DH）が格納されている．変換が成功した場合は，ラベル「ADROUT」にジャンプする

INERR：アドレスバッファに0〜F以外の文字が格納されていた場合は，入力ミスなので，［?］を表示してプログラムの最初に戻り再スタート

CALL CRLFOUT（0152）……復帰/改行を出力

LD A,H〜CALL HEXOUT：HLレジスタペアにセットされているダンプ・アドレスを表示する．1バイトずつ2進1バイト・アスキー16進変換出力ルーチンをコールする

LD A,':'〜CALL CHROUT：［:］とスペースを出力．以上で例えば，（復帰/改行）1234：　という表示となる

* 次のページの注釈を参照

memory data out：HLレジスタペアにセットされているダンプ・アドレスが示すメモリ内容を表示する．以上で1つのアドレスの1バイトのダンプ終了

CALL CHRIN……1文字キー入力

CP 'E'〜JP Z,EXIT：入力がEまたはeならば当プログラムの終了ルーチンへ

CP CR〜JP NZ,START：入力がリターンキー以外ならば，当プログラムの最初から再スタート

```
017D    23                      INC     HL
017E    C3 0152                 JP      ADROUT
                      ;
0181                  ;------ exit this program
                      EXIT:
                                IF      CPM
0181    C3 0000                 JP      0000
                                ENDIF
                      ;
                                IF      PC88
                                RST     38H
                                ENDIF
                      ;
                      ;-------------------------------------
                      ;              subroutine
                      ;-------------------------------------
                      ; ------ this subroutine display 1byte
                      ;        binary data as HEX 8 bit value.
                      ;        A=BINARY data --> display as HEX
0184                  HEXOUT:
0184    F5                      PUSH    AF
0185    0F                      RRCA
0186    0F                      RRCA
0187    0F                      RRCA
0188    0F                      RRCA
0189    CD 018D                 CALL    HEXOU1
018C    F1                      POP     AF
018D                  HEXOU1:
018D    E6 0F                   AND     0FH
018F    C6 30                   ADD     A,30H
0191    FE 3A                   CP      3AH
0193    DA 0198                 JP      C,HEXOU2
0196    C6 07                   ADD     A,7
0198                  HEXOU2:
0198    CD 019C                 CALL    CHROUT
019B    C9                      RET
                      ;
                      ;------ 1 character out subroutine
019C                  CHROUT:
                                IF      CPM
019C    D5                      PUSH    DE
019D    E5                      PUSH    HL
019E    0E 02                   LD      C,2
01A0    5F                      LD      E,A
01A1    CD 0005                 CALL    BDOS
01A4    E1                      POP     HL
01A5    D1                      POP     DE
01A6    C9                      RET
                                ENDIF
```

```
                        ;
                                IF      PC88
                                PUSH    DE
                                PUSH    HL
                                CALL    PCOUT
                                POP     HL
                                POP     DE
                                RET
                                ENDIF
                        ;
                        ;------ carriage return / line feed out subr.
01A7                    CRLFOUT:
01A7    3E 0D                   LD      A,CR
01A9    CD 019C                 CALL    CHROUT
01AC    3E 0A                   LD      A,LF
01AE    CD 019C                 CALL    CHROUT
01B1    C9                      RET
                        ;
                        ;------ 1 character key input subroutine
01B2                    CHRIN:
                                IF      CPM
01B2    C5                      PUSH    BC
01B3    D5                      PUSH    DE
01B4    E5                      PUSH    HL
01B5    0E 01                   LD      C,1
01B7    CD 0005                 CALL    BDOS
01BA    E1                      POP     HL
01BB    D1                      POP     DE
01BC    C1                      POP     BC
01BD    C9                      RET
                                ENDIF
                        ;
                                IF      PC88
                                PUSH    BC
                                PUSH    DE
                                PUSH    HL
                                CALL    PCIN
                                CALL    PCOUT
                                POP     HL
                                POP     DE
                                POP     BC
                                RET
                                ENDIF
                        ;
                        ;-------------------------------------
                        ;       input buffer and stack area
                        ;-------------------------------------
01BE                    ADRBUF  EQU     $
01BE                            DS      5
                        ;
                                IF      CPM
01C3                            DS      32
01E3                    STACK   EQU     $
                                ENDIF
                        ;
                                END     START
```

```
        MACRO-80 3.44    09-Dec-81        PAGE    S

Macros:    アセンブルリストの最後のページには，プログラム全体で
           使われているシンボルの一覧表が示される

Symbols:
01BE    ADRBUF         0152   ADROUT       0127   AHXBI1
0121    AHXBIN         0005   BDOS         01B2   CHRIN
019C    CHROUT         FFFF   CPM          000D   CR
01A7    CRLFOUT        0181   EXIT         0000   FALSE
013D    HCONV          018D   HEXOU1       0198   HEXOU2
0184    HEXOUT         0147   INERR        000A   LF
0110    NEXIN          0000   PC88         3583   PCIN
3E0D    PCOUT          01E3   STACK        0100   START
FFFF    TRUE


No Fatal error(s)
```

では，できあがっている実行可能なオブジェクト・プログラム「DUMPZ.COM」を実行してみましょう．その実行例を次に示します．

**図9-6-3　できあがったオブジェクト・プログラムのCP／M上での実行**

```
A>DUMPZ ↵ ………… 完成したダンプ・プログラムを実行

-0 ………………………… アドレス0000H からダンプ
0000: C3 ↵ ………… アドレス0000H のメモリ内容はC3H．リターンキーの入力により次のアドレスがダンプされる
0001: 03 ↵ ………… アドレス0001H のメモリ内容は03H
0002: DA ↵                 ⋮
0003: 81 ↵                 ⋮
0004: 00x ………………… リターンキー以外を入力すると，再度アドレス入力が可能
-100 ………………………… アドレス0100H を入力
0100: 31 ↵ ………… アドレス0100H のメモリ内容は31H
0101: E3 ↵                 ⋮
0102: 01 ↵                 ⋮
0103: 11 ↵
0104: BEx ……………… リターンキー以外を入力して，再度アドレス入力
-1BE ↵ ……………………… アドレス01BEHを入力．01BEHは当プログラムのダンプ・アドレス・バッファ
01BE: 31 ↵ ……………… 31H =「1」      |
01BF: 42 ↵ ……………… 42H =「B」      | つまり，アドレス01BEHからは，キー入力されたダンプ・アドレスが，
01C0: 45 ↵ ……………… 45H =「E」      | 1，B，E，CRの順で格納されている
01C1: 0Dx ……………… 0DH =キャリジ・リターン」|
-C000 ………………………… 4桁を入力する場合は，↵の入力不要
C000: 00 ↵
C001: 00 ↵
C002: 00x
-E0G0 ………………………… 入力ミスの例．「G」は0～F以外の文字
?………………………………… [?]が表示される
-FFFF
FFFF: 00 ↵ ……………… アドレスFFFFHの次の番地は0000番地
0000: C3e ……………… eまたはEを入力すると当プログラムを終了する

A> …………………………… CP/Mに戻った
```

## BASIC上で実行するオブジェクト・プログラムをM80, L80で作成

次に、PC-8801(mkIIを含む)の $N_{88}$-BASIC上で実行するためのオブジェクト・プログラムを，同じくM80，L80で生成してみましょう．M80アセンブラを実行する前に、条件アセンブルのIF宣言部「CPM」を「FALSE」にしておきます．これにより「PC88」が「NOT CPM」，つまり「TRUE」になり，「PC88」の部分がアセンブルされます．ORG擬似命令の前にASEG擬似命令を挿入しておくことは前項と同じです．

このソース・プログラムのファイル名を「DUMPPC. MAC」として，M80，L80によるアセンブルおよびロードの実行例を示します．

**図9-6-4　M80, L80によるBASIC用オブジェクトの生成**

```
A>DIR DUMPPC.*↲ …………実行前のすべての「DUMPPC」ファイルの確認

A: DUMPPC    MAC
   ソース・プログラムのみ存在

A>M80 DUMPPC,DUMPPC=DUMPPC/Z↲ ………アセンブラM80の実行．最後の「/Z」は，Z-80のニーモニックをア
                                          センブルするためのスイッチ
No Fatal error(s)
アセンブル終了．エセンブル・エラーはなし
A>DIR DUMPPC.*↲ ………アセンブル後の「DUMPPC」ファイルの確認

A: DUMPPC    MAC : DUMPPC    PRN : DUMPPC    REL
                   生成されたアセンブル  生成されたリロケータブ
                   リスト              ル・オブジェクト・プログラム
A>L80 /P:9000,DUMPPC,DUMPPC/N/X/E↲ ………リンクローダL80を実行して，「DUMPPC.REL」から，
           ロード・アドレスを9000Hに指定する     9000Hのロード・アドレスを持ったHEX形式のオブジェ
Link-80  3.44  09-Dec-81  Copyright (c) 1981 Microsoft   クトを生成する．「/X」が
                 ┌─エンド・アドレス                           HEX形式のオブジェク
Data  ┌→9000    90BC    <  188>←プログラム全体のバイト数      トを生成するためのスイ
プログラムのスタート・アドレス                                   ッチ
40830 Bytes Free
[9000   90BC      144]
リンクロード終了

A>DIR DUMPPC.*↲ …………「DUMPPC」ファイルの確認

A: DUMPPC    MAC : DUMPPC    PRN : DUMPPC    REL : DUMPPC    HEX
A>                                               リンクローダにより生成された
                                                 HEX形式のオブジェクト・プログラム
```

この場合も前回と同様に，アセンブルリスト(.PRN)と，オブジェクト・プログラム(.REL)がディスク上に生成されました．ただし今回は，L80リンクローダにより，オブジェクト・プログラム「DUMPPC. REL」から，実行可能な形式のオブジェクト・プログラムではなく，インテルHEX形式のオブジェクト・プログラム「DUMPPC. HEX」を生成しています．つまり，

の変換が行われました．

ではまず，アセンブルリストをタイプアウトしてみましょう．前回は$0100_H$であったスタート・アドレスが，今回は$9000_H$となり，条件アセンブルにより，「PC88」の部分がアセンブルされています．

**図9-6-5　生成されたアセンブルリスト.BASIC用オブジェクトの場合**

```
        MACRO-80 3.44   09-Dec-81          PAGE     1
        解説文は，図9-6-2のCP/M上のプログラムのアセンブルリストも参照

                        ;-------------------------------------------;
                        ;          MEMORY DUMP PROGRAM              ;
                        ;    for [hajimete-yomu-assembler]          ;
                        ;-------------------------------------------;
                        ;
FFFF                    TRUE    EQU     0FFFFH
0000                    FALSE   EQU     NOT TRUE
                        ;
0000                    CPM     EQU     FALSE  ……「偽」
FFFF                    PC88    EQU     NOT CPM ……自動的に「真」になる．N88-BASIC
                                                  上のプログラムを作る
                        ;
0005                    BDOS    EQU     0005H      N88-BASICの1文字キー入力サブ
3583                    PCIN    EQU     3583H ……ルーチンのエントリー・ポイント
3E0D                    PCOUT   EQU     3E0DH ……同1文字出力サブルーチン
000D                    CR      EQU     0DH
000A                    LF      EQU     0AH
                        ;
0000'                           ASEG
                                IF      CPM
                                ORG     100H  ……アセンブルされない
                                ENDIF
                        ;
                                IF      PC88
                                ORG     9000H ……N88BASIC上で実行可能な適
                                ENDIF           当なエリアを指定する
                        ;-------------------------------------------
                        ;             main routine
                        ;-------------------------------------------
9000                    START:
                                IF      CPM
                                LD      SP,STACK ……アセンブルされない
                                ENDIF
                        ;
9000    11 90B7                 LD      DE,ADRBUF
9003    CD 909F                 CALL    CRLFOUT
9006    3E 2D                   LD      A,'-'
9008    CD 9097                 CALL    CHROUT
                        ;
900B    06 04                   LD      B,4
900D                    NEXIN:
900D    CD 90AA                 CALL    CHRIN
9010    12                      LD      (DE),A
9011    FE 0D                   CP      CR
9013    CA 901E                 JP      Z,AHXBIN
9016    13                      INC     DE
9017    05                      DEC     B
9018    C2 900D                 JP      NZ,NEXIN
901B    3E 0D                   LD      A,CR
901D    12                      LD      (DE),A
                        ;
                        ;------ this routine converts ASCII 2byte
                        ;       digits into BINARY HEX.
                        ;   DE=ASCII data pointer  HL=converted data
901E                    AHXBIN:
901E    21 0000                 LD      HL,0
9021    11 90B7                 LD      DE,ADRBUF
```

```
9024                      AHXBI1:
9024    1A                         LD      A,(DE)
9025    FE 0D                      CP      CR
9027    CA 904F                    JP      Z,ADROUT
902A    29                         ADD     HL,HL
902B    29                         ADD     HL,HL
902C    29                         ADD     HL,HL
902D    29                         ADD     HL,HL
902E    CD 903A                    CALL    HCONV
9031    D2 9044                    JP      NC,INERR
9034    85                         ADD     A,L
9035    6F                         LD      L,A
9036    13                         INC     DE
9037    C3 9024                    JP      AHXBI1
903A                      HCONV:
903A    D6 30                      SUB     30H
903C    FE 0A                      CP      0AH
903E    D8                         RET     C
903F    D6 07                      SUB     7
9041    FE 10                      CP      10H
9043    C9                         RET
                          ;
                          ;------ input data is not valid hex value
9044                      INERR:
9044    CD 909F                    CALL    CRLFOUT
9047    3E 3F                      LD      A,'?'
9049    CD 9097                    CALL    CHROUT
904C    C3 9000                    JP      START
                          ;
                          ;------ address out
904F                      ADROUT:
904F    CD 909F                    CALL    CRLFOUT
9052    7C                         LD      A,H
9053    CD 907F                    CALL    HEXOUT
9056    7D                         LD      A,L
9057    CD 907F                    CALL    HEXOUT
905A    3E 3A                      LD      A,':'
905C    CD 9097                    CALL    CHROUT
905F    3E 20                      LD      A,' '
9061    CD 9097                    CALL    CHROUT
                          ;
                          ;------ memory data out
9064    7E                         LD      A,(HL)
9065    CD 907F                    CALL    HEXOUT
                          ;
                          ;------ check continue or new address
9068    CD 90AA                    CALL    CHRIN
906B    FE 45                      CP      'E'
906D    CA 907E                    JP      Z,EXIT
9070    FE 65                      CP      'e'
9072    CA 907E                    JP      Z,EXIT
9075    FE 0D                      CP      CR
9077    C2 9000                    JP      NZ,START
907A    23                         INC     HL
907B    C3 904F                    JP      ADROUT
                          ;
```

```
                        ;------ exit this program
907E                    EXIT:
                                IF      CPM
                                JP      0000        ……アセンブルされない
                                ENDIF                 (CP/M用の当プログラムの終
                        ;                             了処理)
                                IF      PC88
907E      FF                    RST     38H   ……当プログラム終了はリスタート
                                ENDIF                 命令を実行する
                        ;
                        ;-----------------------------------
                        ;             subroutine
                        ;-----------------------------------
                        ; ------ this subroutine display 1byte
                        ;         binary data as HEX 8 bit value.
                        ;        A=BINARY data --> display as HEX
907F                    HEXOUT:
907F      F5                    PUSH    AF
9080      0F                    RRCA
9081      0F                    RRCA
9082      0F                    RRCA
9083      0F                    RRCA
9084      CD 9088               CALL    HEXOU1
9087      F1                    POP     AF
9088                    HEXOU1:
9088      E6 0F                 AND     0FH
908A      C6 30                 ADD     A,30H
908C      FE 3A                 CP      3AH
908E      DA 9093               JP      C,HEXOU2
9091      C6 07                 ADD     A,7
9093                    HEXOU2:
9093      CD 9097               CALL    CHROUT
9096      C9                    RET
                        ;
                        ;------ 1 character out subroutine
9097                    CHROUT:
                                IF      CPM
                                PUSH    DE
                                PUSH    HL
                                LD      C,2
                                LD      E,A         ……アセンブルされない
                                CALL    BDOS          (CP/M用の1文字出力サブル
                                POP     HL            ーチン)
                                POP     DE
                                RET
                                ENDIF
                        ;
                                IF      PC88
9097      D5                    PUSH    DE
9098      E5                    PUSH    HL          N88-BASIC用の1文字出力サブ
9099      CD 3E0D               CALL    PCOUT       ルーチン．Aレジスタにセットされ
909C      E1                    POP     HL          ているデータが表示される
909D      D1                    POP     DE
909E      C9                    RET
                                ENDIF
                        ;
```

```
909F                    ;------ carriage return / line feed out sub r.
909F                    CRLFOUT:
909F     3E 0D                  LD      A,CR
90A1     CD 9097                CALL    CHROUT
90A4     3E 0A                  LD      A,LF
90A6     CD 9097                CALL    CHROUT
90A9     C9                     RET
                        ;
                        ;------ 1 character key input subroutine
90AA                    CHRIN:
                                IF      CPM
                                PUSH    BC
                                PUSH    DE
                                PUSH    HL
                                LD      C,1
                                CALL    BDOS
                                POP     HL
                                POP     DE
                                POP     BC
                                RET
                                ENDIF
                        ;
                                IF      PC88
90AA     C5                     PUSH    BC
90AB     D5                     PUSH    DE
90AC     E5                     PUSH    HL
90AD     CD 3583                CALL    PCIN
90B0     CD 3E0D                CALL    PCOUT
90B3     E1                     POP     HL
90B4     D1                     POP     DE
90B5     C1                     POP     BC
90B6     C9                     RET
                                ENDIF
                        ;
                        ;---------------------------------------
                        ;       input buffer and stack area
                        ;---------------------------------------
90B7                    ADRBUF  EQU     $
90B7                            DS      5
                        ;
                                IF      CPM
                                DS      32
                        STACK   EQU     $
                                ENDIF
                        ;
                                END     START
```

```
          MACRO-80 3.44    09-Dec-81        PAGE     S

Macros:

Symbols:
90B7    ADRBUF          904F    ADROUT          9024    AHXBIT
901E    AHXBIN          0005    BDOS            90AA    CHRIN
9097    CHROUT          0000    CPM             000D    CR
909F    CRLFOUT         907E    EXIT            0000    FALSE
903A    HCONV           9088    HEXOUI          9093    HEXOU2
907F    HEXOUT          9044    INERR           000A    LF
900D    NEXIN           FFFF    PC88            3583    PCIN
3E0D    PCOUT           9000    START           FFFF    TRUE


No Fatal error(s)
```

現在，ディスク上には，9000Hスタートのインテル HEX 形式のオブジェクトファイルができていますので，これを N88-BASIC のディスク上に移し換えれば，5．2章で紹介した，BASIC ベースのアセンブラ開発ツール，DUAD-88D でメモリ上にロードして実行できます．ただし，BASIC プログラムの都合などで，9000Hでは具合が悪い場合は，ソース・プログラムの ORG 指定を他のアドレスに変更し，再アセンブルしてください．

ここでは，できあがったインテル HEX 形式のオブジェクト・プログラムが，PC-8801 の N88-BASIC 上で動作するかどうかのテストが目的なので，簡単に実動テストを行いましょう．次のような方法です．

（1） CP／M上から，DDT を使って，9000Hスタートのインテル HEX 形式のオブジェクト・プログラム「DUMPPC. HEX」をメモリ上にロードする

（2） ディスク・ドライブの電源を切った状態あるいはフロッピーディスクをセットしない状態で，リセットボタンを押す

（3） N88-BASIC が起動しますので，「MON」コマンドでモニタに移り，その「G」（ゴー：実行）コマンドで 9000Hから実行する

では，この一連の手順の実行例を示しましょう．その前に参考として，このオブジェクト・プログラム「DUMPPC. HEX」もタイプアウトしておきます．

図9-6-6　DDTによるHEX形式のオブジェクトのメモリ上へのロード

```
A>TYPE DUMPPC.HEX ↵ …………生成されたHEX形式のオブジェクト・プログラムをタイプアウトする

:2090000011B790CD9F903E2DCD97900604CDAA9012FE0DCA1E901305C20D903E0D12210002
:20902000000011B7901AFE0DCA4F9029292929CD3A90D24490856F13C32490D630FE0AD8D6EF
:209040000007FE10C9CD9F903E3FCD9790C30090CD9F907CCD7F907DCD7F903E3ACD97903E1B
:20906000020CD97907ECD7F90CDAA90FE45CA7E90FE65CA7E90FE0DC2009023C34F90FFF50F
:209080000000F0F0F0FCD8890F1E60FC630FE3ADA9390C607CD9790C9D5E5CD0D3EE1D1C93E89
:1C90A0000DCD97903E0ACD9790C9C5D5E5CD8335CD0D3EE1D1C1C9000000000056
:00900000016F
  ロード・アドレス部                          オブジェクト部   チェックサム部

A>DDT DUMPPC.HEX ↵ ………DDTでHEX形式のオブジェクト・プログラムを，それ自身が持っているロード・
                          アドレス(9000H)に，実行可能なオブジェクト・プログラムに変換してロードする

DDT VERS 2.2
NEXT  PC
90BC 9000

-D8FF0 90BF ↵ …………アドレス8FF0H～90BFHの間をダンプして，プログラムのロード状態を確認
8FF0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
9000 11 B7 90 CD 9F 90 3E 2D CD 97 90 06 04 CD AA 90 ......>-........
9010 12 FE 0D CA 1E 90 13 05 C2 0D 90 3E 0D 12 21 00 ...........>..!.
9020 00 11 B7 90 1A FE 0D CA 4F 90 29 29 29 29 CD 3A ........O.)))).:
9030 90 D2 44 90 85 6F 13 C3 24 90 D6 30 FE 0A D8 D6 ..D..o..$..0....
9040 07 FE 10 C9 CD 9F 90 3E 3F CD 97 90 C3 00 90 CD .......>?.......
9050 9F 90 7C CD 7F 90 7D CD 7F 90 3E 3A CD 97 90 3E ..|...}...>:...>
9060 20 CD 97 90 7E CD 7F 90 CD AA 90 FE 45 CA 7E 90  ...~.......E.~.
9070 FE 65 CA 7E 90 FE 0D C2 00 90 23 C3 4F 90 FF F5 .e.~......#.O...
9080 0F 0F 0F 0F CD 88 90 F1 E6 0F C6 30 FE 3A DA 93 ...........0.:..
9090 90 C6 07 CD 97 90 C9 D5 E5 CD 0D 3E E1 D1 C9 3E ...........>...>
90A0 0D CD 97 90 3E 0A CD 97 90 C9 C5 D5 E5 CD 83 35 ....>..........5
90B0 CD 0D 3E E1 D1 C1 C9 00 00 00 00 00 00 00 00 00 ..>.............
-^C ↵ ……Ctrl-CをキーインしてDDTを終了し，CP/Mに戻る
                                          メモリ上にある実行可能な
A>                                        オブジェクト・プログラム
```

これまでのDDTの操作によって，インテルHEX形式のオブジェクト・プログラムが実行可能なオブジェクト・プログラムの形式で，メモリ上にロードされています(デバッガのDDTやZSIDは，インテルHEX形式のオブジェクトを，自動的に実行可能なオブジェクトに変換する)．

では，ディスク・ドライブの電源を切った状態かあるいはフロッピーディスクをセットしない状態で*リセットボタンを押して，$N_{88}$-BASICを起動し，そのモニタにより，$9000_H$からロードされている$N_{88}$-BASIC用のダンプ・プログラムを実行しましょう．

*フロッピーディスクをセットしたままだと，再びCP/Mが起ち上がってしまう．

**図9-6-7　$N_{88}$-BASIC上でのダンプ・プログラムの実行**

CP/MのDDTでプログラムをメモリ上にロードした後，ディスク・ドライブの電源を切るか，
フロッピーディスクをセットせずに，リセットボタンを押して，N88-BASICを起動する

```
How many files(0-15)?
NEC N-88 BASIC Version 1.0
Copyright (C) 1981 by Microsoft
56799 Bytes free
Ok ……………N88-BASICが起動した
MON ↵ ………モニタに入る

h]D9000 ↵ ………9000H付近をダンプして，ロード状態を確認しておく
9000 11 B7 90 CD 9F 90 3E 2D CD 97 90 06 04 CD AA 90    ｷ┴ﾍﾞ┴>-ﾍ ﾄ  ﾍｴ┴
h]G9000 ↵ ………9000Hからプログラムを実行
-0 ……………………ダンプ・プログラムが実行され，プロンプトの[－]が表示された．ダンプ・アドレス0000Hを入力
0000: F3 ↵ ………アドレス0000Hのメモリ内容はF3H，リターンキーを入力すると，次のアドレスがダンプされる
0001: 31 ↵ ………0001Hは31H
0002: A0x ……………0002HはA0H，リターンキー以外を入力すると，ダンプ・アドレスの再入力となる
-90B7 ……………………アドレス90B7Hは当プログラムのダンプ・アドレス・バッファ
90B7: 39 ↵ ……………90B7Hのメモリ内容は39H＝文字「9」  ┐
90B8: 30 ↵ ……………30H ＝文字「0」                    │ つまりダンプ・アドレス・バッファには
90B9: 42 ↵ ……………42H ＝文字「B」                    │ 9,0,B,7,CRと格納されている
90BA: 37 ↵ ……………37H ＝文字「7」                    │
90BB: 0D ↵ ……………0DH＝キャリジ・リターン・コード    ┘
-C000
C000: 4Fe ……………eまたはEの入力で当プログラムを終了し，モニタに戻る
h] ……………………モニタに戻った
```

いちおう，$N_{88}$-BASIC用に作成されたプログラムは完動することが確認されました．ここでの実行例は，あくまで動作テストだけが目的です．できあがったオブジェクト・プログラムをBASICのディスクや，カセットテープにセーブするには，いろいろな手段があると思いますので試みてください．

# CP/M上で実行するオブジェクト・プログラムをASM, LOADで作成

先の2つのソース・プログラムは，Z-80のニーモニックで書かれていますので，CP/Mのシステムディスクに標準装備されている8080アセンブラ「ASM」では，アセンブルできません．そこで，このダンプ・プログラムを，CP/Mの「ASM」でも開発が可能なように，ソース・プログラムを8080ニーモニックで書き直し，アセンブルした実行例を示しておきます．

前の2つのソース・プログラムは，Z-80アセンブラ用ですが，使っているニーモニックは，すべて8080とコンパチブルのものです．よって，ここで示す8080アセンブラ用のソース・プログラムとは，各ステップごとに1対1で対応しています．当然，アセンブルにより生成されるオブジェクトコードも，ぴったり一致します．両者のアセンブルリストを対比してみてください．

ここで注意しておきたいのは，M80と，ASMの2つのアセンブラでは，アセンブルリストのオブジェクトコードの欄の表現のしかたが，3バイト命令の場合には次のように異なっているということです．

例えば，アドレス$0100_H$のスタック・ポインタ設定命令「LD　SP , STACK」(8080ニーモニックでは「LXI　SP , STACK」)のオブジェクトコード欄は，

| | | アドレス | オブジェクトコード |
|---|---|---|---|
| M80 | の場合 | 0100 | 31　01E3 |
| ASM | の場合 | 0100 | 31E301 |

と表示されています．

つまり，ASMでは，ラベル「STACK」のアドレス，$01E3_H$番地は，メモリ上にロードされる順序どおりに逆にして，"E301"と表示されます．ところが，M80では，そのままの「読む順」に，"01E3"と表示されます．

では，CP/Mの8080アセンブラによるアセンブルと，それにより生成されたインテルHEX形式のオブジェクト・プログラムをローダ「LOAD」により実行可能な純マシン語のオブジェクト・プログラムに変換する，一連の作業の実行例を次のページに示します．

ソース・プログラムは，後で示すアセンブルリストを参照してください．なおこのソース・プログラムのファイル名は，「DUMP80．ASM」として作業をしています．

図9-6-8　CP/Mの8080アセンブラ「ASM」によるアセンブラ

```
A>DIR DUMP80.* ↵ ……………実行前のすべての「DUMP80」ファイルの確認

A: DUMP80    ASM
   ソース・プログラムのみ存在

A>ASM DUMP80 ↵ ……………アセンブラ「ASM」を実行

CP/M ASSEMBLER - VER 2.0
01E3
001H USE FACTOR
END OF ASSEMBLY
アセンブル終了，アセンブル・エラーはなし

A>DIR DUMP80.* ↵ ……………「DUMP80」ファイルの確認

A: DUMP80    ASM : DUMP80    PRN : DUMP80    HEX
                   生成されたアセンブル  生成されたHEX形式
                   リスト              のオブジェクト
A>LOAD DUMP80 ↵ ……………HEX形式のオブジェクトから，実行可能なオブジェクトを生成するローダ「LOAD」の実行

FIRST ADDRESS 0100
LAST  ADDRESS 01BD
BYTES READ    00BE
RECORDS WRITTEN 02
ロード終了

A>DIR DUMP80.* ↵ ……………「DUMP80」ファイルの確認

A: DUMP80    ASM : DUMP80    PRN : DUMP80    HEX : DUMP80    COM
                                                   生成された実行可能な
                                                   オブジェクト・プログラム
A>
```

以上の作業で，CP/Mのシステムディスクに含まれる開発ツールにより，ダンプ・プログラムの実行可能なオブジェクト・プログラム「DUMP80．COM」ができあがりました．これは，図9-6-3のM80，L80で作成されたプログラムの実行例と同じく，「DUMP80 ↵」とキー入力することにより実行されます．次に，この8080用のアセンブルリストを示しておきます．

図9-6-9 8080用ダンプ・プログラムのアセンブルリスト

```
FILE: DUMP80    PRN                        PAGE 001

前回，前々回のプログラムとニーモニックが異なるだけなので，解説文はそれらを参照

                    ;---------------------------------------;
                    ;         MEMORY DUMP PROGRAM           ;
                    ;    for [hajimete-yomu-assembler]      ;
                    ;---------------------------------------;
                    ;
FFFF =              TRUE    EQU     0FFFFH
0000 =              FALSE   EQU     NOT TRUE
                    ;
FFFF =              CPM     EQU     TRUE ···········CP/Mの用プログラムを作る
0000 =              PC88    EQU     NOT CPM
                    ;
0005 =                      BDOS    EQU     0005H
3583 =                      PCIN    EQU     3583H
3E0D =                      PCOUT   EQU     3E0DH
000D =                      CR      EQU     0DH
000A =                      LF      EQU     0AH
                    ;
                            IF      CPM
0100                        ORG     100H
                            ENDIF
                    ;
                            IF      PC88
                            ORG     9000H
                            ENDIF
                    ;---------------------------------------
                    ;             main routine
                    ;---------------------------------------
                    START:
                            IF      CPM
0100 31E301                 LXI     SP,STACK
                            ENDIF
                    ;
0103 11BE01                 LXI     D,ADRBUF
0106 CDA701                 CALL    CRLFOUT
0109 3E2D                   MVI     A,'-'
010B CD9C01                 CALL    CHROUT
                    ;
010E 0604                   MVI     B,4
                    NEXIN:
0110 CDB201                 CALL    CHRIN
0113 12                     STAX    D
0114 FE0D                   CPI     CR
0116 CA2101                 JZ      AHXBIN
0119 13                     INX     D
011A 05                     DCR     B
011B C21001                 JNZ     NEXIN
011E 3E0D                   MVI     A,CR
0120 12                     STAX    D
                    ;
                    ;------ this routine converts ASCII 2byte
                    ;       digits into BINARY HEX.
                    ;   DE=ASCII data pointer  HL=converted data
                    AHXBIN:
0121 210000                 LXI     H,0
0124 11BE01                 LXI     D,ADRBUF
```

```
                AHXBI1:
0127 1A                 LDAX    D
0128 FE0D               CPI     CR
012A CA5201             JZ      ADROUT
012D 29                 DAD     H
012E 29                 DAD     H
012F 29                 DAD     H
0130 29                 DAD     H
0131 CD3D01             CALL    HCONV
0134 D24701             JNC     INERR
0137 85                 ADD     L
0138 6F                 MOV     L,A
0139 13                 INX     D
013A C32701             JMP     AHXBI1
                HCONV:
013D D630               SUI     30H
013F FE0A               CPI     0AH
0141 D8                 RC
0142 D607               SUI     7
0144 FE10               CPI     10H
0146 C9                 RET
                ;
                ;------ input data is not valid hex value
                INERR:
0147 CDA701             CALL    CRLFOUT
014A 3E3F               MVI     A,'?'
014C CD9C01             CALL    CHROUT
014F C30001             JMP     START
                ;
                ;------ address out
                ADROUT:
0152 CDA701             CALL    CRLFOUT
0155 7C                 MOV     A,H
0156 CD8401             CALL    HEXOUT
0159 7D                 MOV     A,L
015A CD8401             CALL    HEXOUT
015D 3E3A               MVI     A,':'
015F CD9C01             CALL    CHROUT
0162 3E20               MVI     A,' '
0164 CD9C01             CALL    CHROUT
                ;
                ;------ memory data out
0167 7E                 MOV     A,M
0168 CD8401             CALL    HEXOUT
                ;
                ;------ check continue or new address
016B CDB201             CALL    CHRIN
016E FE45               CPI     'E'
0170 CA8101             JZ      EXIT
0173 FE65               CPI     'e'
0175 CA8101             JZ      EXIT
0178 FE0D               CPI     CR
017A C20001             JNZ     START
017D 23                 INX     H
017E C35201             JMP     ADROUT
                ;
                ;------ exit this program
```

```
                EXIT:
                        IF      CPM
0181 C30000             JMP     0000
                        ENDIF
                ;
                        IF      PC88
                        RST     7…………Z-80の「RST 38H」は8080では「RST 7」となる
                        ENDIF
                ;
                ;-----------------------------------------
                ;               subroutine
                ;-----------------------------------------
                ; ------ this subroutine display 1byte
                ;        binary data as HEX 8 bit value.
                ;       A=BINARY data --> display as HEX
                HEXOUT:
0184 F5                 PUSH    PSW
0185 0F                 RRC
0186 0F                 RRC
0187 0F                 RRC
0188 0F                 RRC
0189 CD8D01             CALL    HEXOU1
018C F1                 POP     PSW
                HEXOU1:
018D E60F               ANI     0FH
018F C630               ADI     30H
0191 FE3A               CPI     3AH
0193 DA9801             JC      HEXOU2
0196 C607               ADI     7
                HEXOU2:
0198 CD9C01             CALL    CHROUT
019B C9                 RET
                ;
                ;------ 1 character out subroutine
                CHROUT:
                        IF      CPM
019C D5                 PUSH    D
019D E5                 PUSH    H
019E 0E02               MVI     C,2
01A0 5F                 MOV     E,A
01A1 CD0500             CALL    BDOS
01A4 E1                 POP     H
01A5 D1                 POP     D
01A6 C9                 RET
                        ENDIF
                ;
                        IF      PC88
                        PUSH    D
                        PUSH    H
                        CALL    PCOUT
                        POP     H
                        POP     D
                        RET
                        ENDIF
                ;
                ;------ carriage return / line feed out sub r.
                CRLFOUT:
```

```
01A7 3E0D               MVI     A,CR
01A9 CD9C01             CALL    CHROUT
01AC 3E0A               MVI     A,LF
01AE CD9C01             CALL    CHROUT
01B1 C9                 RET
                ;
                ;------ 1 character key input subroutine
                CHRIN:
                        IF CPM
01B2 C5                 PUSH    B
01B3 D5                 PUSH    D
01B4 E5                 PUSH    H
01B5 0E01               MVI     C,1
01B7 CD0500             CALL    BDOS
01BA E1                 POP     H
01BB D1                 POP     D
01BC C1                 POP     B
01BD C9                 RET
                        ENDIF
                ;
                        IF      PC88
                        PUSH    B
                        PUSH    D
                        PUSH    H
                        CALL    PCIN
                        CALL    PCOUT
                        POP     H
                        POP     D
                        POP     B
                        RET
                        ENDIF
                ;
                ;--------------------------------------
                ;       input buffer and stack area
                ;--------------------------------------
01BE =          ADRBUF  EQU     $
01BE                    DS      5
                ;
                        IF      CPM
01C3                    DS      32
01E3 =          STACK   EQU     $
                        ENDIF
                ;
01E3                    END
```

このソース・プログラムのIF宣言部を，$N_{88}$-BASIC用に変更すれば，前節とまったく同様に，BASIC用のオブジェクト・プログラムが作成できます．各自で試みてください．

本来ならば，必ずといってよいほど「デバッグ」作業が必要になります．デバッグについては，次の10章で，本章のオブジェクト・プログラムをデバッグ対象のプログラムとして実習解説していますので，そちらを参照してください．

さて，このコンパクトなダンプ・プログラムの構成内容が理解できれば，これをCP/MデバッガDDTのようなコマンド形式(ダンプ開始アドレス-終了アドレスを指定できるなど)や，表示形式を1行16バイトや，アスキー表示部つきなどに発展させることは，さほど困難ではないでしょう．ぜひ，このプログラムの機能アップに挑戦してみてください．

# 10 デバッガ

4．2章および5．1章でも触れていますが，よほど単純で短いプログラムを除き，最初に作成されたオブジェクト・プログラムには必ずバグがあり，目的どおりの動作をしないものです．このことはどうしても避けられないので，作成したプログラムにはバグがあることを前提に開発を進めなければなりません．

例えば，規模の大きい開発になると，デバッグがしやすいように，あらかじめソース・プログラムの各所にいろいろな「仕掛け」（例えば，要所要所でのレジスタの値や，変数の値などをCRTディスプレイに表示させたりする）を組み込んでおき，バグ退治が完全に終わってから，その仕掛けの部分を外して製品にする，という手段をとります．これには，条件アセンブルのIF～ENDIFの機能を利用するとよいでしょう．

デバッグ作業は，前にも述べたように，アセンブラの知識はもとより，ソフトウェア，ハードウェアに関する総合的な知識と経験が必要になります．ですからデバッグ作業を行うことは，コンピュータ全般の学習，特にアセンブラの学習の面からみて，いろいろな知識が身につくよい機会といえるでしょう．本章は，デバッグについての入門編として，デバッグ作業のテクニックについてではなく，デバッガの基本的な機能を解説します．

では，前章で作成したダンプ・プログラムをデバッグ対象プログラムとして，DDTとZSIDの，2つの代表的デバッガの，基本的な機能を実習解説しましょう．

# 10.1 デバッガが必要とする主な機能

CP/M上の代表的なデバッガには，次に示す「DDT」と「SID」，それに「ZSID」の3つがあります.*

- **DDT** (Dynamic Debugging Tool)—CP/Mのシステムディスクに含まれている8080用デバッガ
- **SID** (Symbolic Instruction Debugger )—ソース・プログラムに使われているシンボル名でアドレスの指定が可能な，DDTの拡張版8080デバッガ
- **ZSID** (Z-80 Symbolic Instruction Debugger)—SIDのZ-80版

これらは，いずれもデジタルリサーチ社の製品であり，この3種のデバッガが，ソフトウェアによる実用的デバッグツールの代表的なものです．デバッグツールには，ICE(インサーキット・エミュレータ)と呼ばれる，ハードウェアを含んだツールもありますが，DDTやZSIDなどは，純ソフトウェアによるツールです．特にZ-80用では，このZSIDが実質的に世界的「標準ツール」であり，マイクロソフト社のM80，L80から出力されるシンボル・テーブル(後述)も，このSID，ZSIDに入力可能な形式に合わせてあります.**

さて，目的どおりに動作しないプログラムをデバッグするには，その誤りの部分を発見するために、各種の機能が必要となります．まず，これらのデ

*5.1章の「デバッガ」の項でも，DDTとZSIDを紹介している

**逆に，リロケータブル・オブジェクト・プログラム形式の実質的標準は，マイクロソフト社のM80が出力する形式であり，デジタルリサーチ社のリロケータブル・マクロアセンブラ「RMAC」は，これに合わせてある．

バッガが持っている機能の代表的なものを挙げてみましょう．(　)内は，それらの機能を実行するときのコマンドで，ここに挙げたものは，DDT，SID，ZSIDのいずれにも共通です．

- **メモリへの任意ファイルのロード(I：インプットおよびR：リード)**
  デバッグしようとするオブジェクト・プログラムや，その他の各種ファイルを，メモリ上の任意のアドレスにロードする

- **ダンプ(D：ダンプ)**
  メモリ内容をダンプする

- **メモリデータの書き換え(S：サブスティテュート)**
  メモリ内容を1バイトごとにダンプし必要があれば書き換える

- **メモリデータのブロック転送(M：ムーブ)**
  メモリの任意のエリアのデータを，任意のアドレスにそっくりコピーする

- **任意のメモリエリア全体を同一パターンで書き込む(F：フィル)**
  メモリの任意のエリアの全体を，任意の1バイトのデータ(00～$FF_H$)で埋めてしまう

- **CPUの各レジスタの状態表示と値のセット(X：イグザミン)**
  CPUの任意のレジスタを，任意の値にセットする

- **ブレーク・ポイントつきのデバッグ対象プログラムの実行(G：ゴー)**
  デバッグ対象プログラムを任意のアドレスから実行する．必要であれば，ブレーク・ポイント(実行を中止するアドレス)を設定する

**• トレース実行（T：トレース）**

デバック対象プログラムを，任意のアドレスから任意のステップだけ実行すると同時に，すべてのステップごとのCPUの全レジスタの状態を表示する

**• 逆アセンブラ（L：リスト）**

メモリ上の任意のアドレスのオブジェクト・プログラムから，その素になるソース・プログラム（ニーモニック）を作り出して表示する

**• ライン・アセンブル（A：アセンブル）**

入力されたニーモニックを1ステップごとにオブジェクトコードに変換し，任意のアドレスから順にメモリに書き込んでいく

以上は，DDTやZSIDの主な機能ですが，一般的なデバッガが必要とする，代表的な基本機能と考えてよいでしょう．

# 10 2 DDTの実行例

まず始めに、DDTや次節のZSIDなどを使って、デバッグ作業が行われている状態のコンピュータのメモリ内の様子を次の図で示しましょう。

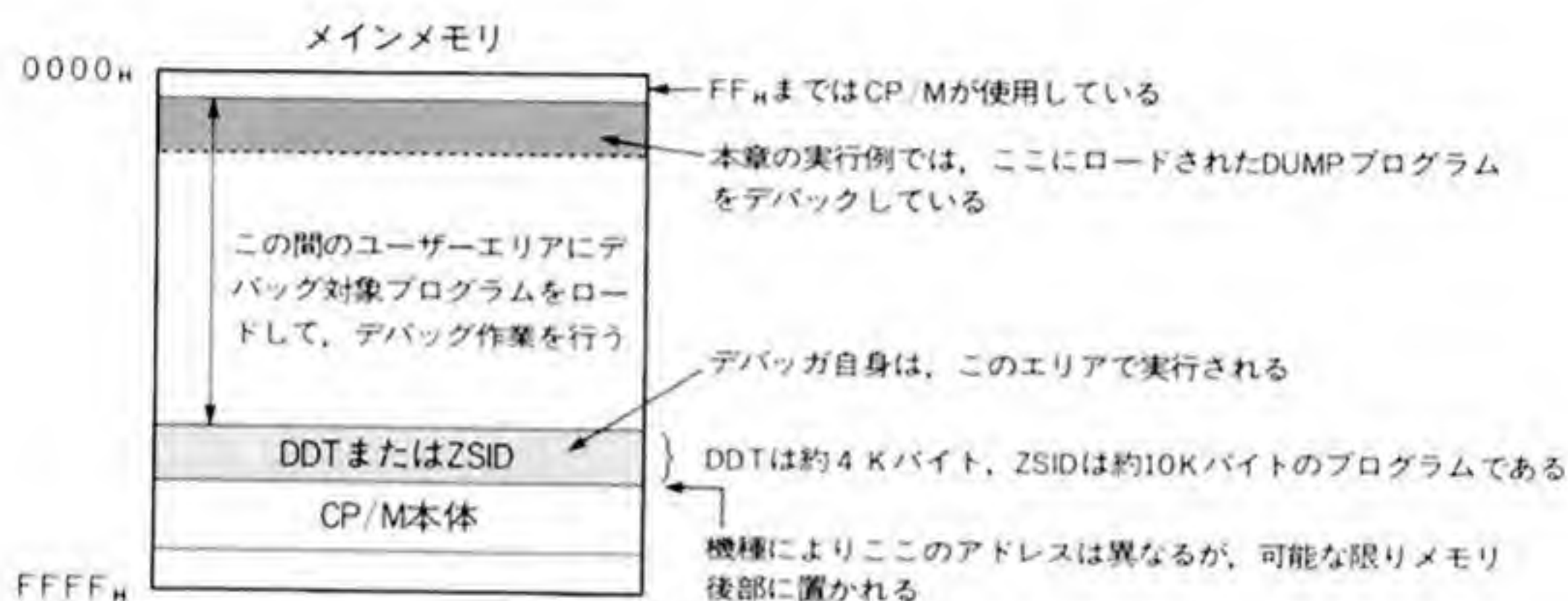



図10-2-1 DDTやZSID実行時のメモリ内の状態

このようにデバッガ自身のプログラムは、デバッグ対象プログラムがロードされるユーザーエリアをなるべく広く連続してとれるように、メモリ後部に位置するCP/M本体の直前に置かれています。

では、前章で作成した簡易版のメモリダンプ・プログラム(スタート・アドレスは$0100_H$)を、デバッグ対象のプログラムとしてメモリにロードし、DDTの持ついくつかの機能を実行してみましょう。ただしここでは、DDTのコマンドの使い方や、デバッグ作業のテクニックを解説するのではなく、デバッガが持つ基本的機能を紹介することを第一の目的としています。

前章では、8080用とZ-80用の同じ内容のダンプ・プログラムのソースから、CP/Mの「ASM」と、マイクロソフト社の「M80」のそれぞれのアセンブラにより、2つのインテルHEX形式のオブジェクト・プログラムが作成されています。それらのファイル名はDUMP80.HEXとDUMPZ.HEXの2つです。

この2つのオブジェクト・プログラムは，8080とZ-80とのコンパチブルのニーモニックのみを使ったソース・プログラムから生成されていますので，中身はまったく同じです．よって，DDTのデバッグ対象プログラムとしてどちらを使うこともできますが，ここでは「DUMP80．HEX」を使います．

ではまず，このオブジェクト・プログラム「DUMP80．HEX」の内容をタイプアウトしておきましょう．

**図10-2-2　DDTでデバッグされるオブジェクト「DUMP80．HEX」**

```
A>TYPE DUMP80.HEX
:1001000031E30111BE01CDA7013E2DCD9C010604B6
:10011000CDB20112FE0DCA21011305C210013E0D20
:100120001221000011BE011AFE0DCA52012929290F
:1001300029CD3D01D24701856F13C32701D630FE7B
:1001400000AD8D607FE10C9CDA7013E3FCD9C01C3FA
:10015000001CDA7017CCD84017DCD84013E3ACD47
:100160009C013E20CD9C017ECD8401CDB201FE4597
:10017000CA8101FE65CA8101FE0DC2000123C3527E
:1001800001C30000F50F0F0F0FCD8D01F1E60FC673
:1001900030FE3ADA9801C607CD9C01C9D5E50E02BA
:1001A0005FCD0500E1D1C93E0DCD9C013E0ACD9C3D
:0E01B000001C9C5D5E50E01CD0500E1D1C1C9DB
:0000000000
A>
```



次に，このインテルHEX形式のオブジェクト・プログラムを，DDTのロード機能により実行可能な純マシン語のオブジェクト・プログラムに変換してメモリ上にロードし，DDTのいくつかの機能を実行してみましょう．

ここでは特に，1バイトのメモリ内容を，CRTディスプレイへ16進数として表示するサブルーチンである「2進1バイト ⇨ アスキー16進変換出力」(ラベル「HEXOUT：」)の部分に注目し，その変換処理の開始から表示の完了までの全ステップを，「トレース実行」します．よって，その過程のCPUの動きをすべて見ることができますので，このプログラムのアセンブルリスト(前章の**図9－6－2**，および**図9－6－9**)と対比し，ステップごとの各レジスタの変化をよく観察してください．*

*DDTは，8080CPU用のデバッガなので，トレースや逆アセンブラなどのリストは，8080ニーモニックで表示される．前述の図9-6-2のZ-80でのアセンブルリストと対比してみるとよい．

図10-2-3　DDTの各種機能の実行例

```
A>DDT DUMP80.HEX ↵ ………DDTを起動して，HEX形式のデバッグ対象プログラムのオブジェクトを，それ自
                          身が持つロード・アドレス(0100H)に，実行可能なオブジェクトに変換してロードする
DDT VERS 2.2
NEXT  PC
01BE 0000
DDT起動，デバッグ対象プログラムのロード終了
-D100 1BF ↵ ………ダンプ・プログラムがロードされたエリア，0100H～01BFHの間をダンプする(D:ダンプコマンド)
0100 31 E3 01 11 BE 01 CD A7 01 3E 2D CD 9C 01 06 04 1.........>-.....
0110 CD B2 01 12 FE 0D CA 21 01 13 05 C2 10 01 3E 0D .......!......>.
0120 12 21 00 00 11 BE 01 1A FE 0D CA 52 01 29 29 29 .!.........R.)))
0130 29 CD 3D 01 D2 47 01 85 6F 13 C3 27 01 D6 30 FE ).=..G..o..'..0.
0140 0A D8 D6 07 FE 10 C9 CD A7 01 3E 3F CD 9C 01 C3 ..........>?....
0150 00 01 CD A7 01 7C CD 84 01 7D CD 84 01 3E 3A CD .....|...}...>:.
0160 9C 01 3E 20 CD 9C 01 7E CD 84 01 CD B2 01 FE 45 ..> ...~.......E
0170 CA 81 01 FE 65 CA 81 01 FE 0D C2 00 01 23 C3 52 ....e........#.R
0180 01 C3 00 00 F5 0F 0F 0F 0F CD 8D 01 F1 E6 0F C6 ................
0190 30 FE 3A DA 98 01 C6 07 CD 9C 01 C9 D5 E5 0E 02 0.:.............
01A0 5F CD 05 00 E1 D1 C9 3E 0D CD 9C 01 3E 0A CD 9C _......>....>...
01B0 01 C9 C5 D5 E5 0E 01 CD 05 00 E1 D1 C1 C9 13 CD ................
```

この後に5バイトのダンプ・アドレス・バッファと，32バイトのスタックエリアが続くが，そのオブジェクトは必要ないので生成されていない

```
-L184 19B ↵ ………アドレス0184H～019BHの間を逆アセンブル，8080用の
                  ニーモニックで表示される(L:リストコマンド)
  0184  PUSH PSW
  0185  RRC
  0186  RRC
  0187  RRC
  0188  RRC
  0189  CALL 018D
  018C  POP  PSW
  018D  ANI  0F
  018F  ADI  30
  0191  CPI  3A
  0193  JC   0198
  0196  ADI  07
  0198  CALL 019C
  019B  RET
  019C
```

ソースプログラムと比較してみること
ただし〰〰部はラベルではなく，絶対アドレス値で表示される

```
-G100 184 ↵ ………アドレス0184H(ラベルHEXOUT:)にブレーク・ポイントを設定して，0100Hから実行をスタート(G:ゴーコマンド)
   ←── DDTのプロンプトではなく，実行されたプログラムによるプロンプトである(同じ[-]でまぎらわしいが)
-AB01 ………デバッグ対象プログラムであるダンプ・プログラムが起動して，プロンプト[-]が表示されたのでAB01とキー入力した
*0184 ………アドレス0184Hに設定したブレーク・ポイントで実行が停止した

-X ↵ ………そのときのCPUの各レジスタの状態を見る(X:イグザミンコマンド)
C0Z1M0E1I0 A=AB B=000A D=01C2 H=AB01 S=01E1 P=0184 PUSH PSW
```

C,Z,M,E,Iの各フラグの状態　Aレジスタ　BCレジスタ　DEレジスタ　HLレジスタ　スタック・ポインタ　プログラム・カウンタ　現在の命令

```
-D1B0 1CF ↵ ………ダンプ・アドレス・バッファ付近をダンプする
01B0 01 C9 C5 D5 E5 0E 01 CD 05 00 E1 D1 C1 C9 41 42 ..............AB
01C0 30 31 0D 21 6A 1E 70 2B 71 2A 69 1E EB 0E 14 CD 01.!j.p+q*i.....
```

入力されたアドレス「AB01」が，アスキーコードでこのようにメモリ上には格納されている

```
-TC↵……現在の状態(プログラム・カウンタ=0184H)から,CH=12ステップ分のトレース実行を行う(T:トレースコマンド)
C0Z1M0E1I0 A=AB B=000A D=01C2 H=AB01 S=01E1 P=0184 PUSH PSW
C0Z1M0E1I0 A=AB B=000A D=01C2 H=AB01 S=01DF P=0185 RRC
C1Z1M0E1I0 A=D5 B=000A D=01C2 H=AB01 S=01DF P=0186 RRC
C1Z1M0E1I0 A=EA B=000A D=01C2 H=AB01 S=01DF P=0187 RRC
C0Z1M0E1I0 A=75 B=000A D                 S=01DF P=0188 RRC
C1Z1M0E1I0 A=BA B=000A D これらのレジスタは, S=01DF P=0189 CALL 018D
C1Z1M0E1I0 A=BA B=000A D ここでは直接関係な S=01DD P=018D ANI  0F
C0Z0M0E1I1 A=0A B=000A D いので無視してよい S=01DD P=018F ADI  30
C0Z0M0E0I0 A=3A B=000A D                 S=01DD P=0191 CPI  3A
C0Z1M0E0I0 A=3A B=000A D=01C2 H=AB01 S=01DD P=0193 JC   0198
C0Z1M0E0I0 A=3A B=000A D=01C2 H=AB01 S=01DD P=0196 ADI  07
C0Z0M0E0I1 A=41 B=000A D=01C2 H=AB01 S=01DD P=0198 CALL 019C*019C
  各フラグ   Aレジスタ                         プログラム・ CPU命令,8080ニーモニック
                                               カウンタ     で表示される

 2進1バイト→アスキー16進2バイト変換出力サブルーチン「HEXOUT」の前半部.Aレジスタのデータ ABH の上位
 の「A」を文字としてスクリーンに表示する部分をトレース実行したもの.この部分ではAレジスタのデータ ABH の
 「A」が,アスキーコード41Hに変換されて同じAレジスタにセットされるまでの過程が示されている.CPU命令に
 対するAレジスタおよびキャリーフラグ「C」の変化に注目

-G,18C↵ ……Aレジスタに41Hがセットされている状態で,アドレス018CHにブレーク・ポイントを置いて実行,1文字出力ルーチンが実行される
A*018C……018CHで実行が停止した
└─ スクリーンに表示された「A」つまりABHのAが表示された.ダンプ・アドレス(AB01)表示の最初の文字の表示
-X↵ …………前出
C0Z1M0E1I0 A=00 B=0041 D=01C2 H=AB01 S=01DF P=018C POP  PSW
(ここは,2進1バイトのデータABHのAがアスキー16進2バイト変換ルーチンの途中で出力された時点である)
-T7↵ …………ここから7ステップ分のトレース実行
C0Z1M0E1I0 A=00 B=0041 D=01C2 H=AB01 S=01DF P=018C POP  PSW
C0Z1M0E1I0 A=AB B=0041 D=01C2 H=AB01 S=01E1 P=018D ANI  0F
C0Z0M0E0I1 A=0B B=0041 D=01C2 H=AB01 S=01E1 P=018F ADI  30
C0Z0M0E0I0 A=3B B=0041 D=01C2 H=AB01 S=01E1 P=0191 CPI  3A
C0Z0M0E0I0 A=3B B=0041 D=01C2 H=AB01 S=01E1 P=0193 JC   0198
C0Z0M0E0I0 A=3B B=0041 D=01C2 H=AB01 S=01E1 P=0196 ADI  07
C0Z0M0E0I1 A=42 B=0041 D=01C2 H=AB01 S=01E1 P=0198 CALL 019C*019C

 2進1バイト→アスキー16進2バイト変換出力の後半部.Aレジスタのデータ ABH の「B」が,
 アスキーコードの42Hに変換されて同じAレジスタにセットされるまでの過程が示されている

                 Aレジスタに42Hがセットされている状態で,当プログラムの終了ルーチンであるアドレス0181H
-G,181↵ ……にブレーク・ポイントを置いて実行,Eまたはeが入力されるまで実行が続く
B01: 0D↵ ……まず「B」が表示され,さらにプログラムの実行が続き,AB01番地のダンプが終了する.
              「A」はすでに表示されているので,ここではBから始まっている
AB02: 0D↵ ┐
AB03: 0D↵ ┘↵の入力により,次のアドレスに進む
AB04: 0D. ……↵以外の入力により,新しいダンプ・アドレスの入力となる
-1BE↵ ……………次はアドレス01BEHをダンプする(ダンプ・アドレス・バッファである).ここの[-]はDDTのものではない
01BE: 31↵ ┐
01BF: 42↵ │31H=「1」,42H=「B」,45H=「E」である
01C0: 45↵ ┘
01C1: 0De*0181 ……eの入力により終了ルーチンに進み,ブレーク・ポイント0181Hで実行が停止した
再びDDTに戻った
-D100 10F↵ ……………DDTによるダンプ(特に意味はない)
0100 31 E3 01 11 BE 01 CD A7 01 3E 2D CD 9C 01 06 04 1........>-.....

-^C …………Ctrl-Cの入力によりDDTを終了する

A>…………CP/Mに戻った
```

実行例の中の，「G」コマンドにより，任意のアドレスからプログラムを実行していますが，このときに，2つの「ブレーク・ポイント」を設定しておくことができます．ブレーク・ポイントとは，実行をとりやめる(ブレークする)アドレスのことであり，このアドレスにくると，そこでプログラムの実行が中止され，デバッガ(この場合はDDT)にコントロールが戻ります．ブレークした時点のCPUの各レジスタの状態を見れば，いろいろなことが分析できるでしょう．

このブレーク・ポイントは，最も重要なデバッグの機能のひとつで，この機能をうまく利用することが，デバッグ・テクニックのポイントです．

# 10 3 SID, ZSIDの機能と実行例

シンボリック・インストラクション・デバッガのSIDやZSIDは，その名のとおり「シンボル名」を使ってデバッグすることが可能なデバッガです．前項のDDTでは，その実行例に示されているように，アドレスの指定を絶対値で行いました．例えば，ラベル「HEXOUT：」のアドレスであれば「0184」などと絶対値を入力しなければならないわけです．

これに対して，シンボリック・インストラクション・デバッガは，ソース・プログラムに使われているシンボル名(各種のシンボルや，ラベルなど)を使って，それらがつけられているアドレス値を指定することができます．つまり，ラベル「HEXOUT：」であれば，「0184」と入力しなくても，「．HEXOUT」と入力すればよいのです(先頭に[．]をつけることに注意)．もちろん絶対値で入力してもかまいません．

ただし，シンボルを使ったデバッグを可能にするための前提としては，シンボル・テーブル・ファイル(シンボル名とそのアドレス値の一覧表．後ほど実際のものをリストで示す)が必要であり，このテーブルをあらかじめZSIDに入力しておかなければなりません．シンボル・テーブル・ファイルは，デジタルリサーチ社のアセンブラであるMACやRMACの場合は，アセンブラの実行により，自動的に生成されますが，M80の場合は，次の手続きが必要です．

M80, L80の場合は，L80実行時に／Yスイッチをつけることにより，ソース・プログラムの「パブリック・シンボル」* として宣言されているシンボルのシンボルテーブルが作成されます．よって，デバッグに必要なシンボルは，ソース・プログラムの段階で，パブリック・シンボルとして宣言しておくとよいでしょう．

* 他のモジュールでも使用可能なシンボル．

パブリック・シンボルなどの解説は，次の11章で行いますが，ここではその宣言の方法のひとつである，ソース・プログラムのラベルにつけるコロン[：]を，2重の[：：]にする方法を使って，パブリックであることを宣言しています．例えば，

HEXOUT：　⇨　HEXOUT：：

とするわけです．なお，パブリック宣言がされていないものはシンボルテーブルに登録されませんので，そのシンボルを使ってデバッグすることはできません．

ここで，M80，L80により，シンボル・テーブル・ファイルを作成する作業の流れを図示しておきましょう．

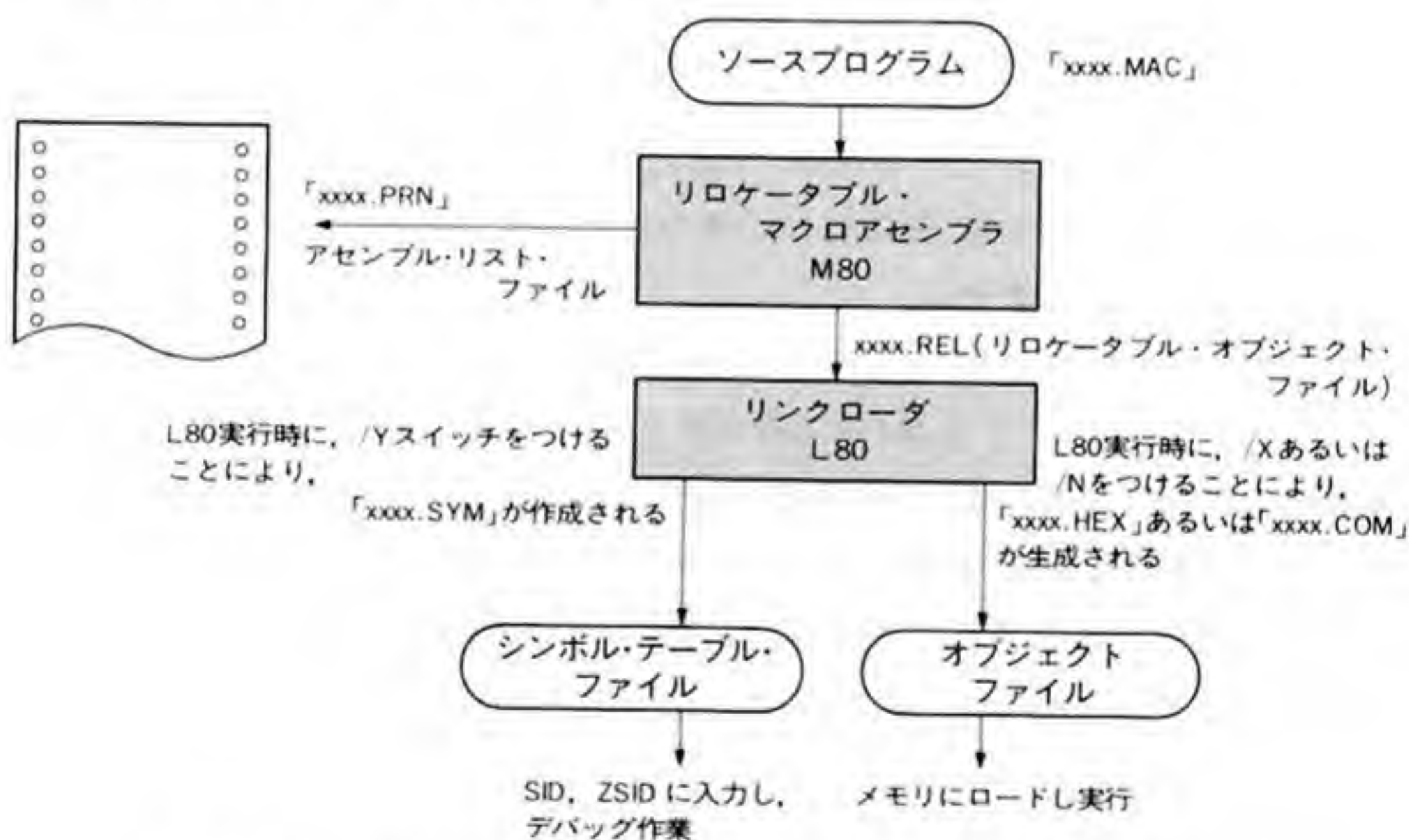



**図10-3-1　シンボルテーブル・ファイルが作成される作業の流れ**

では次に，シンボルテーブルを得るために，前章のダンプ・プログラムのラベルをパブリック宣言したソース・プログラムを，M80によるアセンブルリストで示します．M80は，8080，Z-80の両方で利用できるアセンブラですので，8080用，Z-80用のどちらのソース・プログラムを使って実習してもかまいません．

### 図10-3-2　ダンプ・プログラムのソースのラベルをパブリック宣言したもの

```
        MACRO-80 3.44   09-Dec-81       PAGE    1


                                ;--------------------------------------;
                                ;         MEMORY DUMP PROGRAM          ;
                                ;    for [hajimete-yomu-assembler]     ;
                                ;--------------------------------------;
                                        .Z80
                                ;
FFFF                            TRUE    EQU     0FFFFH
0000                            FALSE.  EQU     NOT TRUE
                                ;
FFFF                            CPM     EQU     TRUE
0000                            PC88    EQU     NOT CPM
                                ;
0005                            BDOS    EQU     0005H
3583                            PCIN    EQU     3583H
3E0D                            PCOUT   EQU     3E0DH
000D                            CR      EQU     0DH
000A                            LF      EQU     0AH
                                ;
0000                                    ASEG
                                ;
                                        IF      CPM
                                        ORG     100H
                                        ENDIF
                                ;
                                        IF      PC88
                                        ORG     9000H
                                        ENDIF
                                ;---------------------------------------
                                ;          main routine
                                ;---------------------------------------
0100                            START::
                                        IF      CPM
0100    31 01E3                         LD      SP,STACK
                                        ENDIF
                                ;
0103    11 01BE                         LD      DE,ADRBUF
0106    CD 01A7                         CALL    CRLFOUT
0109    3E 2D                           LD      A,'-'
010B    CD 019C                         CALL    CHROUT
                                ;
010E    06 04                           LD      B,4
0110                            NEXIN::
0110    CD 01B2                         CALL    CHRIN
0113    12                              LD      (DE),A
0114    FE 0D                           CP      CR
0116    CA 0121                         JP      Z,AHXBIN
0119    13                              INC     DE
011A    05                              DEC     B
011B    C2 0110                         JP      NZ,NEXIN
011E    3E 0D                           LD      A,CR
0120    12                              LD      (DE),A
                                ;
                                ;------ this routine converts ASCII 2byte
                                ;       digits into BINARY HEX.
                                ;   DE=ASCII data pointer  HL=converted data
0121                            AHXBIN::
```

.Z80 …… 当ソース・プログラムが，Z-80ニーモニックであることを宣言する擬似命令．これがあるとM80実行時の/Zスイッチが不要となる

TRUE …… CP/M上のプログラムを作る

```
0121    21 0000                 LD      HL,0
0124    11 01BE                 LD      DE,ADRBUF
0127                    AHXBI1::
0127    1A                      LD      A,(DE)
0128    FE 0D                   CP      CR
012A    CA 0152                 JP      Z,ADROUT
012D    29                      ADD     HL,HL
012E    29                      ADD     HL,HL
012F    29                      ADD     HL,HL
0130    29                      ADD     HL,HL
0131    CD 013D                 CALL    HCONV
0134    D2 0147                 JP      NC,INERR
0137    85                      ADD     A,L
0138    6F                      LD      L,A
0139    13                      INC     DE
013A    C3 0127                 JP      AHXBI1
013D                    HCONV::
013D    D6 30                   SUB     30H
013F    FE 0A                   CP      0AH
0141    D8                      RET     C
0142    D6 07                   SUB     7
0144    FE 10                   CP      10H
0146    C9                      RET
                        ;
                        ;------ input data is not valid hex value
0147                    INERR::
0147    CD 01A7                 CALL    CRLFOUT
014A    3E 3F                   LD      A,'?'
014C    CD 019C                 CALL    CHROUT
014F    C3 0100                 JP      START
                        ;
                        ;------ address out
0152                    ADROUT::
0152    CD 01A7                 CALL    CRLFOUT
0155    7C                      LD      A,H
0156    CD 0184                 CALL    HEXOUT
0159    7D                      LD      A,L
015A    CD 0184                 CALL    HEXOUT
015D    3E 3A                   LD      A,':'
015F    CD 019C                 CALL    CHROUT
0162    3E 20                   LD      A,' '
0164    CD 019C                 CALL    CHROUT
                        ;
                        ;------ memory data out
0167    7E                      LD      A,(HL)
0168    CD 0184                 CALL    HEXOUT
                        ;
                        ;------ check continue or new address
016B    CD 01B2                 CALL    CHRIN
016E    FE 45                   CP      'E'
0170    CA 0181                 JP      Z,EXIT
0173    FE 65                   CP      'e'
0175    CA 0181                 JP      Z,EXIT
0178    FE 0D                   CP      CR
017A    C2 0100                 JP      NZ,START
017D    23                      INC     HL
```

```
017E    C3 0152                 JP      ADROUT
                        ;
                        ;------ exit this program
0181                    EXIT::
                                IF      CPM
0181    C3 0000                 JP      0000
                                ENDIF
                        ;
                                IF      PC88
                                RST     38H
                                ENDIF
                        ;
                        ;--------------------------------------
                        ;               subroutine
                        ;--------------------------------------
                        ; ------ this subroutine display 1byte
                        ;         binary data as HEX 8 bit value.

                        ;       A=BINARY data --> display as HEX
0184                    HEXOUT::
0184    F5                      PUSH    AF
0185    0F                      RRCA
0186    0F                      RRCA
0187    0F                      RRCA
0188    0F                      RRCA
0189    CD 018D                 CALL    HEXOU1
018C    F1                      POP     AF
018D                    HEXOU1::
018D    E6 0F                   AND     0FH
018F    C6 30                   ADD     A,30H
0191    FE 3A                   CP      3AH
0193    DA 0198                 JP      C,HEXOU2
0196    C6 07                   ADD     A,7
0198                    HEXOU2::
0198    CD 019C                 CALL    CHROUT
019B    C9                      RET
                        ;
                        ;------ 1 character out subroutine
019C                    CHROUT::
                                IF      CPM
019C    D5                      PUSH    DE
019D    E5                      PUSH    HL
019E    0E 02                   LD      C,2
01A0    5F                      LD      E,A
01A1    CD 0005                 CALL    BDOS
01A4    E1                      POP     HL
01A5    D1                      POP     DE
01A6    C9                      RET
                                ENDIF

                                IF      PC88
                                PUSH    DE
                                PUSH    HL
                                CALL    PCOUT
                                POP     HL
                                POP     DE
                                RET
```

```
                                ENDIF
                        ;
                        ;------ carriage return / line feed out subr.
01A7                    CRLFOUT::
01A7    3E 0D                   LD      A,CR
01A9    CD 019C                 CALL    CHROUT
01AC    3E 0A                   LD      A,LF
01AE    CD 019C                 CALL    CHROUT
01B1    C9                      RET
                        ;
                        ;------ 1 character key input subroutine
01B2                    CHRIN::
                                IF      CPM
01B2    C5                      PUSH    BC
01B3    D5                      PUSH    DE
01B4    E5                      PUSH    HL
01B5    0E 01                   LD      C,1
01B7    CD 0005                 CALL    BDOS
01BA    E1                      POP     HL
01BB    D1                      POP     DE
01BC    C1                      POP     BC
01BD    C9                      RET
                                ENDIF

                                IF      PC88
                                PUSH    BC
                                PUSH    DE
                                PUSH    HL
                                CALL    PCIN
                                CALL    PCOUT
                                POP     HL
                                POP     DE
                                POP     BC
                                RET
                                ENDIF
                        ;
                        ;------------------------------------------
                        ;       input buffer and stack area
                        ;------------------------------------------
01BE                    ADRBUF  EQU     $
01BE                            DS      5
                        ;
                                IF      CPM
01C3                            DS      32
01E3                    STACK   EQU     $
                                ENDIF
                        ;
                                END     START
```

```
        MACRO-80 3.44   09-Dec-81        PAGE    S

                これはアセンブルリストのシンボル一覧表であり，SID,ZSIDに
Macros:         入力するためのシンボル・テーブルではない

Symbols:
01BE    ADRBUF          01521   ADROUT          01271   AHXBI1
01211   AHXBIN          0005    BDOS            01B21   CHRIN
019C1   CHROUT          FFFF    CPM             000D    CR
01A71   CRLFOUT         01811   EXIT            0000    FALSE
013D1   HCONV           018D1   HEXOU1          01981   HEXOU2
01841   HEXOUT          01471   INERR           000A    LF
01101   NEXIN           0000    PC88            3583    PCIN
3E0D    PCOUT           01E3    STACK           01001   START
FFFF    TRUE


No Fatal error(s)
```

次に，このアセンブルリストで示したソース・プログラム(ファイル名は，DUMP．MAC)をM80でアセンブルし，L80でインテルHEX形式のオブジェクト・プログラムと，シンボルテーブルを生成する作業の実行例を示します．

図10-3-3 M80，L80によるHEX形式のオブジェクトとシンボルテーブルの作成

```
A>DIR DUMP.*⏎ …………実行前の「DUMP」ファイルの確認

A: DUMP     MAC
   ソース・プログラムのみ存在

A>M80 DUMP,DUMP=DUMP⏎ ………M80によるアセンブル，ソース・プログラムはZ-80ニーモニックであるが，今回は
                              /Zスイッチは不要(ソース・プログラムの冒頭部の「.Z80」擬似命令による)
No Fatal error(s)
アセンブル終了．アセンブル・エラーはなし
A>DIR DUMP.*⏎ ………「DUMP」ファイルの確認

A: DUMP     MAC : DUMP     PRN : DUMP     REL
                  生成された       生成されたリロケータブル・
                  アセンブルリスト  オブジェクト・ファイル
A>L80 DUMP,DUMP/N/Y/X/E⏎ ………L80によるリンクローダの実行，/Yによりシンボル・テーブル，/Xに
                                 よりHEX形式のオブジェクト・プログラムが生成される
Link-80  3.44  09-Dec-81  Copyright (c) 1981 Microsoft

Data    0100    01E3    <  227>

40791 Bytes Free
[0100   01E3        1]
リンクロード終了
A>DIR DUMP.*⏎ ………「DUMP」ファイルの確認
A: DUMP     MAC : DUMP     PRN : DUMP     REL : DUMP     SYM
A: DUMP     HEX
                                                生成されたシンボル・
   生成されたHEX形式                             テーブル・ファイル
A> のオブジェクト・プログラム
```

ここまでの作業で，ダンプ・プログラムのインテルHEX形式のオブジェクト・プログラム「DUMP．HEX」と，そのシンボルテーブル「DUMP．SYM」のファイルが，ディスク上に生成されています．このいずれもアスキーファイルですので，そのままタイプアウトして確認してみましょう．

図10-3-4　生成されたオブジェクトとシンボルテーブルのファイル内容の確認

```
A>TYPE DUMP.HEX ↵ ……生成されたHEX形式のオブジェクト・プログラムをタイプアウトする
                                                                  ┌─オブジェクト部
:2001000031E30111BE01CDA7013E2DCD9C010604CDB20112FE0DCA21011305C210013E0DE7
:200120001221000011BE011AFE0DCA520129292929CD3D01D24701856F13C32701D630FEBB
:200140000AD8D607FE10C9CDA7013E3FCD9C01C30001CDA7017CCD84017DCD84013E3ACD92
:200160009C013E20CD9C017ECD8401CDB201FE45CA8101FE65CA8101FE0DC2000123C35286
:2001800001C30000F50F0F0F0FCD8D01F1E60FC630FE3ADA9801C607CD9C01C9D5E50E02BE
:2001A0005FCD0500E1D1C93E0DCD9C013E0ACD9C01C9C5D5E50E01CD0500E1D1C1C90000C7
:2001C0000000000000000000000000000000000000000000000000000000000000000000001F
:0301E0000000001C
:00010001FE                    └─5バイトのダンプ・アドレス・バッファ+32バイトのスタックエリア

A>TYPE DUMP.SYM ↵ …………生成されたシンボル・テーブルをタイプアウトする

0152 ADROUT      0127 AHXBI1      0121 AHXBIN      01B2 CHRIN
019C CHROUT      01A7 CRLF0D      0181 EXIT        013D HCONV
018D HEXOU1      0198 HEXOU2      0184 HEXOUT      0147 INERR
0110 NEXIN       0100 START

A>          ソース・プログラムでパブリック宣言されているシンボルが一覧表になっている
```

このように，ソース・プログラムに[：：]をつけてパブリック宣言したラベルのシンボル・テーブル・ファイル「DUMP．SYM」が作成されています．このシンボルテーブルを，デバッグ対象プログラムであるダンプ・プログラムのオブジェクト・プログラム「DUMP．HEX」とともに，ZSIDでロードすることにより，シンボリックなデバッグが可能になります．

では，シンボリック・インストラクション・デバッガ，ZSIDの代表的な機能のいくつかを実行してみましょう．ZSID内のコマンドの使い方は，DDTと同じです．

ここでは特に，「アスキー16進4桁 ⇨ 2進2バイト変換」のルーチン（ラベル「AHXBIN：：」）の部分に注目し，その変換処理の開始から表示の完了までの全ステップを，トレース実行します．その間のCPUの動きをすべて見ることができますので，ステップごとの，各レジスタの変化や変換の過程をよく観察してください．

**図10-3-5　ZSIDによるシンボリックなデバッグ作業**

```
A>ZSID↵ ……… ZSIDを起動する

ZSID VERS 1.4
ZSIDが起動した．ZSIDのプロンプトは「#」
#IDUMP.HEX DUMP.SYM↵ ……… HEX形式のオブジェクト・プログラムと，シンボル・テーブルを
                            メモリ上にロードする準備をする(I：インプットコマンド)
#R↵ ……… ディスクからメモリ上に実際にロードする(R：リードコマンド)

SYMBOLS
NEXT  PC   END     シンボル・テーブルと，デバッグ対象プログラムであるダンプ・プログラムが，
01E3 0100 A988     実行可能なオブジェクトに変換されて，メモリにロードされた

#D100 1FF↵ ……… ロードされたオブジェクト・プログラムをダンプして確認(D：ダンプコマンド)
0100: 31 E3 01 11 BE 01 CD A7 01 3E 2D CD 9C 01 06 04
      1  .  .  .  .  .  .  .  .  >  -  .  .  .  .  .
0110: CD B2 01 12 FE 0D CA 21 01 13 05 C2 10 01 3E 0D
      .  .  .  .  .  .  .  !  .  .  .  .  .  .  >  .
0120: 12 21 00 00 11 BE 01 1A FE 0D CA 52 01 29 29 29
      .  !  .  .  .  .  .  .  .  .  .  R  .  )  )  )
                       (
                       (       DDTではこの位置に表示されたアスキー表示は，左の16進表示の
                       (       各バイトの下に表示されている

01A0: 5F CD 05 00 E1 D1 C9 3E 0D CD 9C 01 3E 0A CD 9C
      _  .  .  .  .  .  .  >  .  .  .  .  >  .  .  .
01B0: 01 C9 C5 D5 E5 0E 01 CD 05 00 E1 D1 C1 C9 00 00   ~~~部の5バイトはダンプ・
      .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .   アドレス・バッファ，これ以降
01C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   01E2Hまではスタックエリア
      .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .

#L100 120↵ ……… アドレス0100H～0120Hを逆アセンブルする(L：リストコマンド)
START: ←
  0100  LD   SP,01E3
  0103  LD   DE,01BE
  0106  CALL 01A7 .CRLFOU ←
  0109  LD   A,2D
  010B  CALL 019C .CHROUT ←
  010E  LD   B,04
NEXIN: ←
  0110  CALL 01B2 .CHRIN ←
  0113  LD   (DE),A
  0114  CP   0D
  0116  JP   Z,0121 .AHXBIN ←
  0119  INC  DE
  011A  DEC  B
  011B  JP   NZ,0110 .NEXIN ←
  011E  LD   A,0D
  0120  LD   (DE),A         ←印のように，ラベルが表示されている．例えばCALL 01A7 .CRLFOU
AHXBIN: ←                   は，01A7HがラベルCRLFOUであることを示している．また，ZSIDは当然
  0121                                  ながらZ-80のニーモニックで表示される

#D.CRLFOU .CHRIN↵ ……ダンプの範囲をラベルで指定できる．ラベルCRLFOU：からCHRIN：の間をダンプ．ピリオド「.」を必ずつけること
01A7: 3E 0D CD 9C 01 3E 0A CD 9C
      >  .  .  .  .  >  .  .  .
01B0: 01 C9 C5
      .  .  .
```

```
#L..CRLFOU .CHRIN ↵ ………逆アセンブルする範囲をラベルで指定する。ラベルCRLFOU：からCHRIN：間を逆アセンブル
CRLFOU:
  01A7  LD   A,0D
  01A9  CALL 019C .CHROUT
  01AC  LD   A,0A
  01AE  CALL 019C .CHROUT
  01B1  RET
CHRIN:
  01B2  PUSH BC
  01B3 ………現在の各レジスタやフラグの状態を調べる(X：イグザミンコマンド)
#X ↵ …Aレジスタ BCレジスタ DEレジスタ HLレジスタ スタック・ポインタ プログラム・カウンタ
 ----- A=00 B=0000 D=0000 H=0000 S=0100 P=0100 ………表レジスタとスタック・ポインタおよびプログラム・カウンタの状態
 ----- A'00 B'0000 D'0000 H'0000 X=0000 Y=0000 LD   SP,01E3 …裏レジスタとIXおよびIYレジスタの状態と、そのときのCPU命令
C,Z,M,E,Iの各フラグの状態。「-」は"0"を表す(裏レジスタはA'，B'のように「'」をつけて表レジスタと区別している)
#G100 .AHXBIN ↵ …ブレーク・ポイントをラベルAHXBIN：のアドレスに設定して，0100Hから実行(G：ゴーコマンド)

-1234 ………DUMPプログラムが実行され，そのプロンプト「-」が表示された。それに続いて，ダンプ・アドレスを1234と入力した

*0121 .AHXBIN ………アドレス0121H(ラベルAHXBIN：)で実行が停止した。つまり，ダンプ・アドレス1234Hを入力したため，入力されたアドレスデータを2進2バイトに変換するルーチンに実行が進み，ここでブレーク・ポイントに出合った
#D1BE 1C2 ↵ ………今回は絶対アドレスでダンプの範囲を指定。ここはダンプ・アドレス・バッファのエリアにあたる
01BE: 31 32
      1  2      入力したダンプ・アドレス1234Hが，このようにアスキーコードで格納されている。31H="1"，32H="2"……
01C0: 33 34 0D
      3  4
………フラグ名が表示されているものは"1"の状態
#X ↵ ………現在の各レジスタおよびフラグの状態の確認
 -Z--- A=0D B=002D D=01C2 H=0000 S=01E3 P=0121
 ----- A'00 B'0000 D'0000 H'0000 X=0000 Y=0000 LD   HL,0000
#T45 ↵ ………45Hステップ分トレース実行する。裏レジスタは，当プログラムの実行には何ら関係しないので，以降のリスト上では削除してある
 -Z--- A=0D B=002D D=01C2 H=0000 S=01E3 P=0121 LD   HL,0000
 -Z--- A=0D B=002D D=01C2 H=0000 S=01E3 P=0124 LD   DE,01BE
                                                     ダンプ・アドレス・バッファの先頭アドレス
AHXBI1:
 -Z--- A=0D B=002D D=01BE H=0000 S=01E3 P=0127 LD   A,(DE) ………その1文字目をAレジスタにセット
 -Z--- A=31 B=002D D=01BE H=0000 S=01E3 P=0128 CP   0D
 ----I A=31 B=002D D=01BE H=0000 S=01E3 P=012A JP   Z,0152 .ADROUT
 ----I A=31 B=002D D=01BE H=0000 S=01E3 P=012D ADD  HL,HL
 ----- A=31 B=002D D=01BE H=0000 S=01E3 P=012E ADD  HL,HL
 ----- A=31 B=002D D=01BE H=0000 S=01E3 P=012F ADD  HL,HL
 ----- A=31 B=002D D=01BE H=0000 S=01E3 P=0130 ADD  HL,HL
 ----- A=31 B=002D D=01BE H=0000 S=01E3 P=0131 CALL 013D .HCONV
HCONV:
 ----- A=31 B=002D D=01BE H=0000 S=01E1 P=013D SUB  30
 ----- A=01 B=002D D=01BE H=0000 S=01E1 P=013F CP   0A
 C-M-I A=01 B=002D D=01BE H=0000 S=01E1 P=0141 RET  C
 C-M-I A=01 B=002D D=01BE H=0000 S=01E3 P=0134 JP   NC,0147 .INERR
 C-M-I A=01 B=002D D=01BE H=0000 S=01E3 P=0137 ADD  A,L
 ----- A=01 B=002D D=01BE H=0000 S=01E3 P=0138 LD   L,A
 ----- A=01 B=002D D=01BE H=0001 S=01E3 P=0139 INC  DE
 ----- A=01 B=002D D=01BF H=0001 S=01E3 P=013A JP   0127 .AHXBI1
AHXBI1:
 ----- A=01 B=002D D=01BF H=0001 S=01E3 P=0127 LD   A,(DE) ………2文字目をAレジスタにセット
 ----- A=32 B=002D D=01BF H=0001 S=01E3 P=0128 CP   0D
 ----I A=32 B=002D D=01BF H=0001 S=01E3 P=012A JP   Z,0152 .ADROUT
 ----I A=32 B=002D D=01BF H=0001 S=01E3 P=012D ADD  HL,HL
 ----- A=32 B=002D D=01BF H=0002 S=01E3 P=012E ADD  HL,HL
 ----- A=32 B=002D D=01BF H=0004 S=01E3 P=012F ADD  HL,HL
 ----- A=32 B=002D D=01BF H=0008 S=01E3 P=0130 ADD  HL,HL
 ----- A=32 B=002D D=01BF H=0010 S=01E3 P=0131 CALL 013D .HCONV
```

```
HCONV:
 -----  A=32  B=002D D=01BF H=0010 S=01E1 P=013D SUB  30
 -----  A=02  B=002D D=01BF H=0010 S=01E1 P=013F CP   0A
 C-M-I  A=02  B=002D D=01BF H=0010 S=01E1 P=0141 RET  C
 C-M-I  A=02  B=002D D=01BF H=0010 S=01E3 P=0134 JP   NC,0147  .INERR
 C-M-I  A=02  B=002D D=01BF H=0010 S=01E3 P=0137 ADD  A,L
 -----  A=12  B=002D D=01BF H=0010 S=01E3 P=0138 LD   L,A
 -----  A=12  B=002D D=01BF H=0012 S=01E3 P=0139 INC  DE
 -----  A=12  B=002D D=01C0 H=0012 S=01E3 P=013A JP   0127  .AHXBI1
AHXBI1:
 -----  A=12  B=002D D=01C0 H=0012 S=01E3 P=0127 LD   A,(DE)  ……3文字目をAレジスタにセット
 -----  A=33  B=002D D=01C0 H=0012 S=01E3 P=0128 CP   0D
 ----I  A=33  B=002D D=01C0 H=0012 S=01E3 P=012A JP   Z,0152  .ADROUT
 ----I  A=33  B=002D D=01C0 H=0012 S=01E3 P=012D ADD  HL,HL
 -----  A=33  B=002D D=01C0 H=0024 S=01E3 P=012E ADD  HL,HL
 -----  A=33  B=002D D=01C0 H=0048 S=01E3 P=012F ADD  HL,HL
 -----  A=33  B=002D D=01C0 H=0090 S=01E3 P=0130 ADD  HL,HL
 -----  A=33  B=002D D=01C0 H=0120 S=01E3 P=0131 CALL 013D  .HCONV
HCONV:
 -----  A=33  B=002D D=01C0 H=0120 S=01E1 P=013D SUB  30
 -----  A=03  B=002D D=01C0 H=0120 S=01E1 P=013F CP   0A
 C-M-I  A=03  B=002D D=01C0 H=0120 S=01E1 P=0141 RET  C
 C-M-I  A=03  B=002D D=01C0 H=0120 S=01E3 P=0134 JP   NC,0147  .INERR
 C-M-I  A=03  B=002D D=01C0 H=0120 S=01E3 P=0137 ADD  A,L
 -----  A=23  B=002D D=01C0 H=0120 S=01E3 P=0138 LD   L,A
 -----  A=23  B=002D D=01C0 H=0123 S=01E3 P=0139 INC  DE
 -----  A=23  B=002D D=01C1 H=0123 S=01E3 P=013A JP   0127  .AHXBI1
AHXBI1:
 -----  A=23  B=002D D=01C1 H=0123 S=01E3 P=0127 LD   A,(DE)  ……4文字目をAレジスタにセット
 -----  A=34  B=002D D=01C1 H=0123 S=01E3 P=0128 CP   0D
 ----I  A=34  B=002D D=01C1 H=0123 S=01E3 P=012A JP   Z,0152  .ADROUT
 ----I  A=34  B=002D D=01C1 H=0123 S=01E3 P=012D ADD  HL,HL
 -----  A=34  B=002D D=01C1 H=0246 S=01E3 P=012E ADD  HL,HL
 -----  A=34  B=002D D=01C1 H=048C S=01E3 P=012F ADD  HL,HL
 -----  A=34  B=002D D=01C1 H=0918 S=01E3 P=0130 ADD  HL,HL
 ----I  A=34  B=002D D=01C1 H=1230 S=01E3 P=0131 CALL 013D  .HCONV
HCONV:
 ----I  A=34  B=002D D=01C1 H=1230 S=01E1 P=013D SUB  30
 -----  A=04  B=002D D=01C1 H=1230 S=01E1 P=013F CP   0A
 C-M-I  A=04  B=002D D=01C1 H=1230 S=01E1 P=0141 RET  C
 C-M-I  A=04  B=002D D=01C1 H=1230 S=01E3 P=0134 JP   NC,0147  .INERR
 C-M-I  A=04  B=002D D=01C1 H=1230 S=01E3 P=0137 ADD  A,L
 -----  A=34  B=002D D=01C1 H=1230 S=01E3 P=0138 LD   L,A
 -----  A=34  B=002D D=01C1 H=1234 S=01E3 P=0139 INC  DE
 -----  A=34  B=002D D=01C2 H=1234 S=01E3 P=013A JP   0127  .AHXBI1
AHXBI1:
 -----  A=34  B=002D D=01C2 H=1234 S=01E3 P=0127 LD   A,(DE)  ……5文字目をAレジスタにセット
 -----  A=0D  B=002D D=01C2 H=1234 S=01E3 P=0128 CP   0D
 -Z---  A=0D  B=002D D=01C2 H=1234 S=01E3 P=012A JP   Z,0152  .ADROUT
                             ←── 2進2バイトへの変換完了
*0152  .ADROUT ……アドレス0152H(ラベルADROUT:)でブレークがかかり，実行が停止した

#G,.START ……ブレーク・ポイントをラベルSTART:に設定して，現在のアドレス(プログラム・カウンタ)から実行する

1234:  16x ……ダンプ・プログラムの実行による表示．ダンプ・アドレス1234Hの1バイトがダンプされた後，リターンキー以外の入力を行った

*0100  .START ……何らかのキーの入力により，新しいダンプ・アドレスの入力となるために，再びプログラムの先頭に戻り，そこでブレークがかかった

#D1234 1234↵ ……ZSIDに戻っている．ZSIDによるアドレス1234Hのダンプ．先のDUMPプログラムによる
1234:  16          ダンプと対比させただけで，特に意味はない
```

```
#G,.EXIT ↵ ··············· 先ほどのアドレス(ラベルSTART:)から、ブレーク・ポイントをラベル「EXIT」に設定して

-1BE ↵ ·················· DUMPプログラムが実行されたので、ダンプ・アドレス01BEHを入力した
01BE: 31↵ ┐
01BF: 42↵ │ リターンキーにより、1バイトずつダンプされていく
01C0: 45↵ ┘
01C1: 0De ················ eの入力によりDUMPプログラムを終了する

*0181 .EXIT ·············· ラベル「EXIT:」でブレークがかかった
#^C ······················ Ctrl-Cの入力によりZSIDを終わり、CP/Mへ戻る
A> ······················· CP/Mに戻った
```

# 11 リロケータブルマクロアセンブラの概念と使い方

リロケータブル・マクロアセンブラは，アセンブラによるソフトウェア開発の実務になくてはならないツールであり，実務用のアセンブラといえば，このリロケータブル・マクロアセンブラを指すといってもよいでしょう．

リロケータブル・マクロアセンブラとは，リロケータブル機能とマクロ機能との両方を持ったアセンブラです．本章では，その機能の中で，リロケータブル機能について解説します．リロケータブル機能を利用すると，モジュール別のソフトウェア開発が可能になります．ある程度の規模の開発になると，ソース・プログラムが何百ステップ，何千ステップにもなり，それを1本のソース・プログラムで書き上げることは種々の弊害があって現実的ではありません．そこで，よほど小さなプログラムを除き，ある程度のプログラムの開発には，全体のプログラムをいくつかの機能別のブロック(モジュール)に分割して，それぞれを独立したソース・プログラムとして開発し，最終的に1本のオブジェクト・プログラムにまとめる「モジュール別ソフトウェア開発法」を採ります．

本章では，このモジュール別ソフトウェア開発法の実際を実習する意味で，9章で作成したダンプ・プログラムをいくつかのモジュールのソース・プログラムに分割し，モジュール別の開発を行います．ただし，これはあくまで実習が目的であり，本来はこのような小さなプログラムに対して，モジュール別のソフトウェア開発を行うのは意味がありません．なお，リロケータブル・マクロアセンブラについては，5.1章でも触れていますので，そちらも参照してください．

# 11-1 リロケータブル マクロアセンブラの概念

リロケータブル・マクロアセンブラは，リロケータブル・アセンブラの機能と，マクロアセンブラの機能を合わせ持ったアセンブラです．いずれも高度な内容を含んだアセンブラであり，特にマクロアセンブラの機能は，これを解説するだけで，1冊の本になってしまいます．そこで本書では，リロケータブル・マクロアセンブラの機能の中でも，リロケータブル機能を利用したモジュール別ソフトウェア開発法を中心に，その概要を解説しましょう．

モジュール別ソフトウェア開発法の必要性を理解するために，もし，その方法を採らずに1本の長いソース・プログラムを作成すると，どのような弊害が起こるかを挙げてみましょう．

- 1本の長大なプログラムでは，その構成の管理が非常に困難である
- 1本のソース・プログラムでは，そのすべてが作成されてからでないと，アセンブルすることができない
- ひとつのプログラムが非常に大きいので，アセンブル→デバッグ→アセンブル，の繰返しの作業に手間がかかる
- 大きなプログラムになると，複数のプログラマーが分担して開発を進めることになるが，全員が1本のソース・プログラムを作成するのは，作業の行程，進行の管理上の問題が多い

こうした弊害に対してモジュール別開発法には次に示す利点があります．

- プログラム全体を，機能別のソース・プログラム単位で構造的に分割できるので，階層的なプログラミングの管理が容易である
- プログラム全体を，各モジュール単位で開発できるので，他のモジュールの影響が少なく，開発能率がよい

- 開発がモジュール単位なので，作業の行程，進行状況の管理が容易である
- デバッグ作業において，修正を行うモジュールが直接関連あるものだけに限られ，他のモジュールに影響しないので，能率的である

これらのことから，小さなプログラムの開発は別として，それ以外にはリロケータブル・マクロアセンブラを使わなければ実務的ではありません．このアセンブラの代表が「M80」あるいは「RMAC」であるわけです．

次に，1本のソース・プログラムで開発を行う場合と，モジュール別のソフトウェア開発を行う場合との，2つの作業の流れを図示しましょう．この図の中のファイル名やモジュール名などは，次節の実行例のものですが，ここでは何か大きなプログラムのファイル名と思ってください．

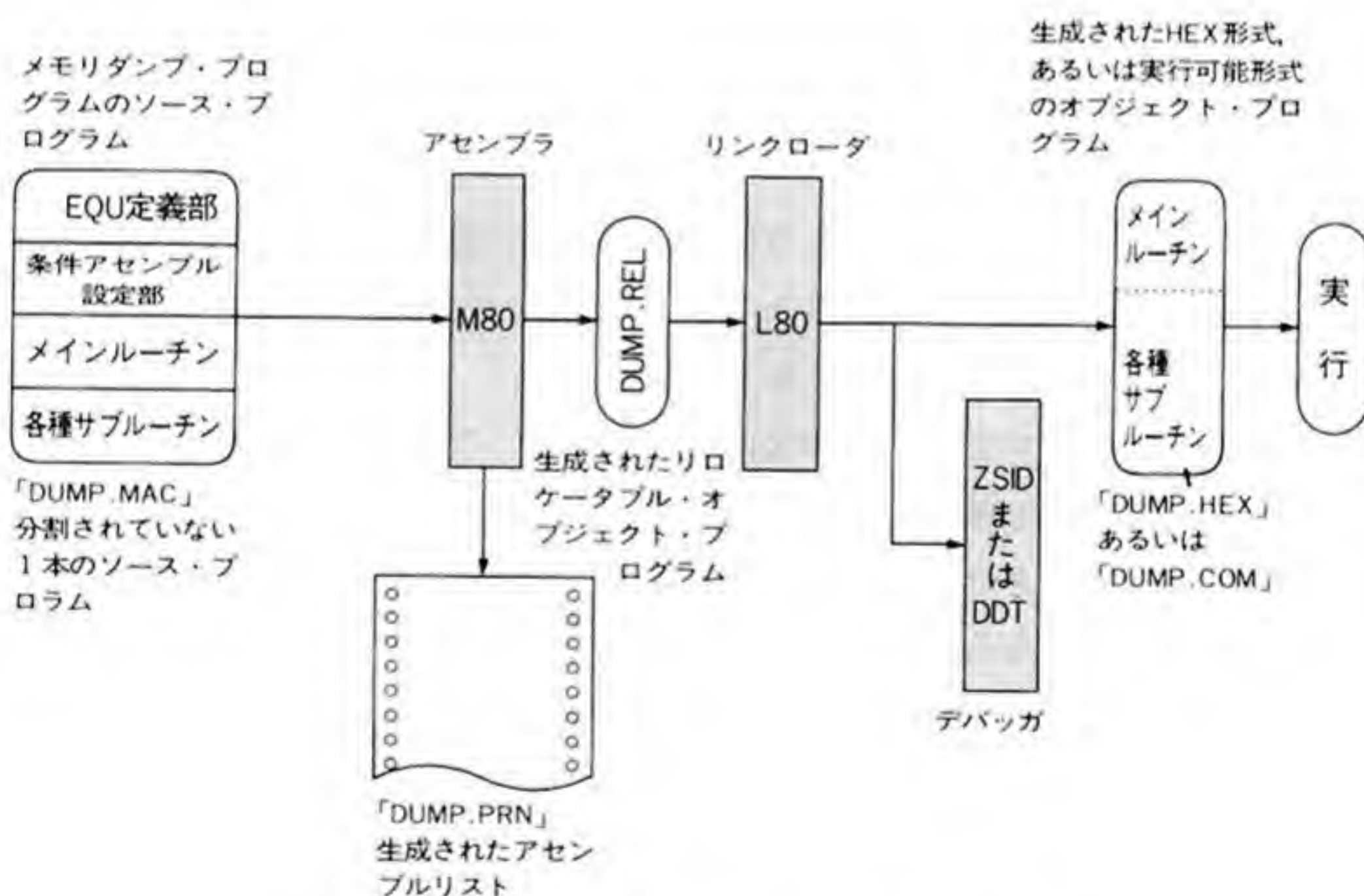



図11-1-1　1本のソース・プログラムでの開発作業

この手順は，今までの実行例で行ってきたもので，特に解説しなくてもよいでしょう．これに使用するアセンブラが，アブソリュート・アセンブラの場合は，ロード・アドレスの固定されたオブジェクト・プログラムが生成されますが，リロケータブル・アセンブラであれば，生成されたオブジェクト・プログラムに対して，ロード・アドレスを任意に設定することが可能です．

次の図は，モジュール別ソフトウェア開発法の手順です．

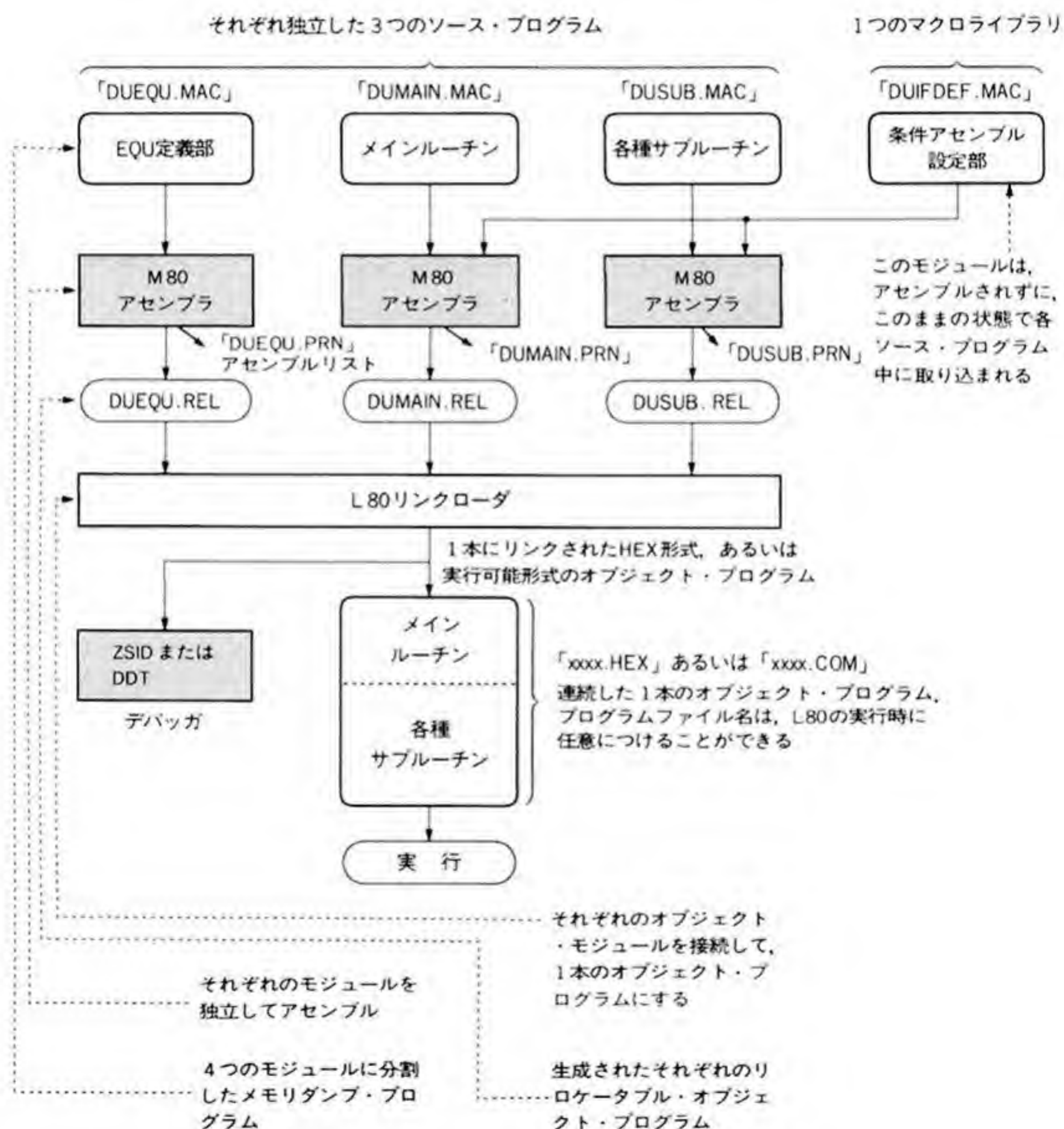



**図11-1-2　モジュール別のソフトウェア開発手順**

この図の場合は，1つのプログラムを開発するために，これを4つのモジュール(ブロック)に分け，それぞれを別々のソース・プログラムとして独立に開発しています．それぞれのモジュールのソース・プログラム(xxxx．MAC)がアセンブルされると，それに対するリロケータブル・オブジェクト・プログラム(xxxx．REL)が個々に生成されます．

ここが重要なところで，ここに「リロケータブル・オブジェクト・プログラム」の必要性があるのです．つまり，それぞれのモジュールが，独立して開発されているわけですから，互いに他のモジュールが占めるオブジェクト・プログラムの大きさ，つまりバイト数は不明です．先頭モジュールのアドレスは，指定するのでよいとして，それに続くモジュールのロード・アドレスは，各モジュールのオブジェクト・プログラムができあがって，それぞれを接続する段階でなくてはわからないわけです．そこで，各モジュールの接続を，それぞれに正しいロード・アドレスを与えながら行う「リンクローダ」が登場するわけです．

ここで，もしオブジェクト・プログラムの形式が，リロケータブル形式ではなく，絶対アドレスを持ったオブジェクト・プログラムであれば，モジュール別のソフトウェア開発がいかに困難になるかを考えてみると，リロケータブル・アセンブラの概念がさらに明確になるでしょう．

# 11/2 モジュール別ソフトウェア開発法の実習解説

モジュール別のソフトウェア開発を行う場合は，それぞれのモジュール間で，互いのシンボル(サブルーチンなども含まれる)を，共同で使ったりします．この場合，通常のアセンブラでは，ソース・プログラム上にないシンボルやサブルーチンを使えば，当然アセンブル・エラーとなります．そこでリロケータブル・アセンブラには，「PUBLIC(パブリック)」と「EXTRN(エクスターナル)」の2つの擬似命令が用意されており，これによって別のソース・プログラム中のシンボルを使う(外部のシンボルを参照する)ことなどを宣言することによって，独立したモジュール間のシンボルを相互に関係づけることができます．これらは次のように使います．

| | |
|---|---|
| PUBLIC　　ABC | ――「ABC」というシンボルを他のモジュールで使用してもよいことを宣言する |
| EXTRN　　ABC | ――別のモジュール上のシンボル「ABC」を使用することを宣言する |

このPUBLICとEXTRNの概念を，次の図で示しましょう．

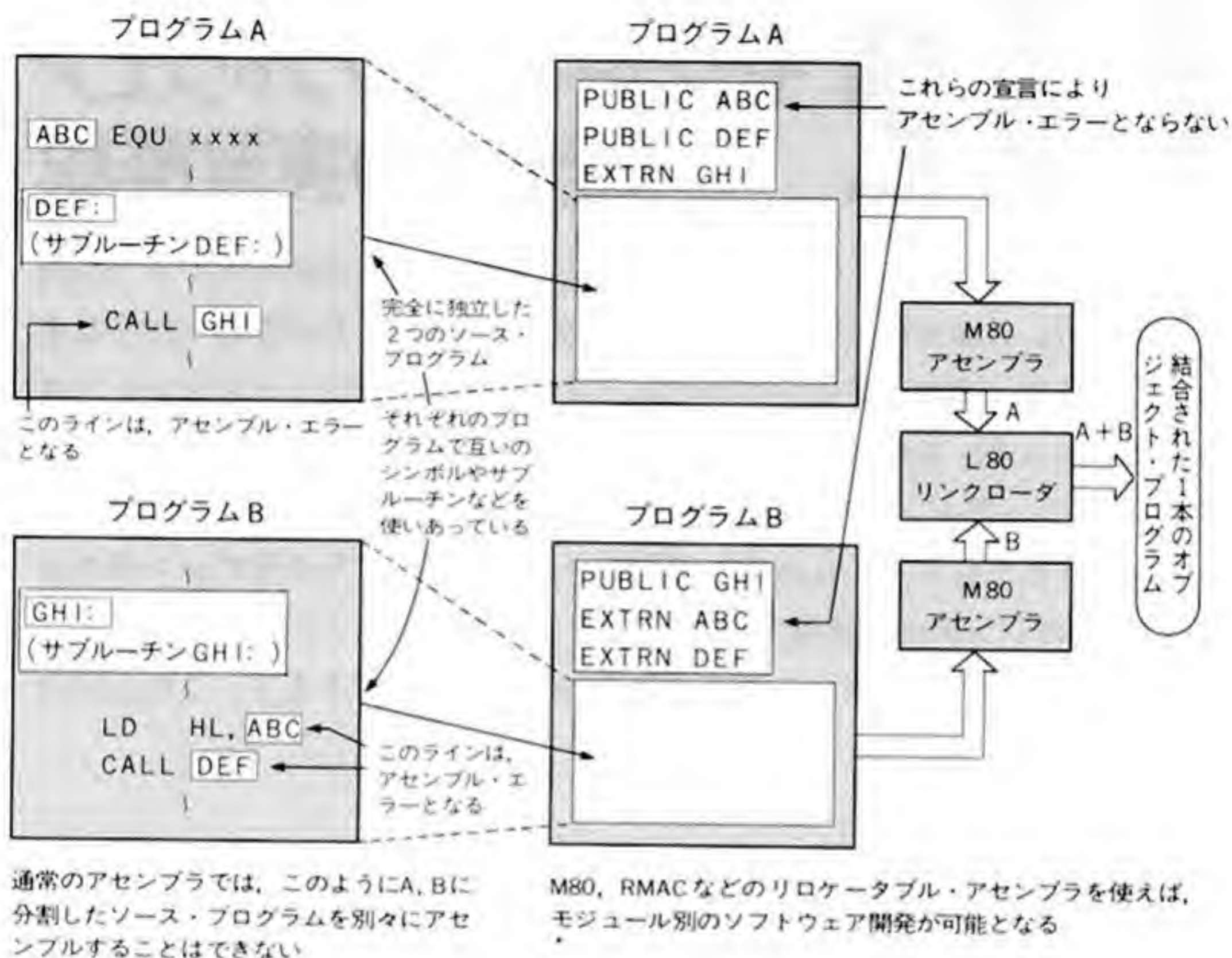



**図11-2-1 PUBLIC, EXTRNの考え方**

擬似命令「PUBLIC」，および「EXTRN」を使用するかわりに，シンボルやラベルに直接[::]および[##]の記号を付加することによっても，同様に宣言することができます．その様子を次の図で示します．[::]については，10．3章の実行例でも使用していますので参照してください．

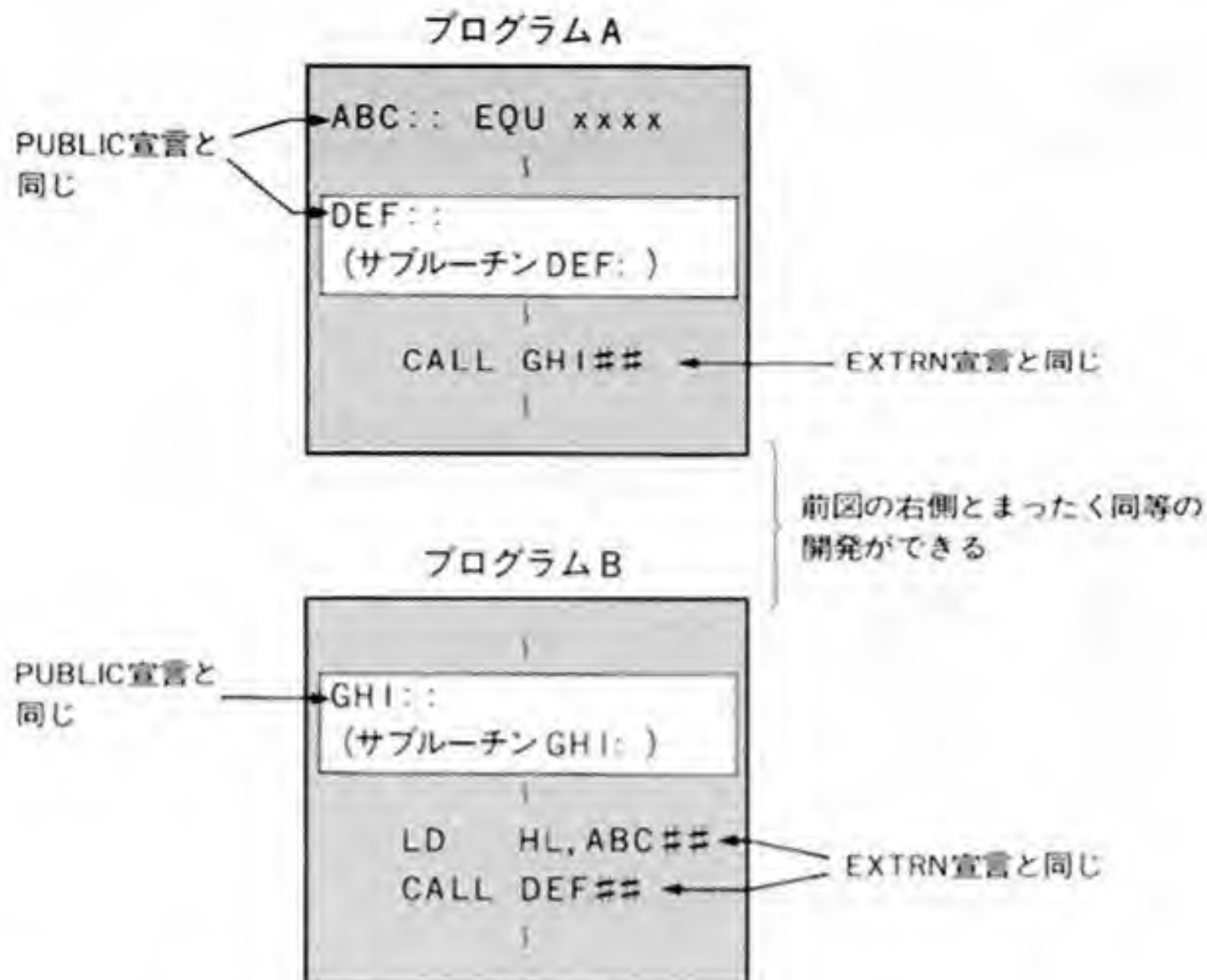



**図11-2-2 PUBLIC, EXTRNの[::], [##]による直接宣言**

この2つの擬似命令の使い方が理解できれば、モジュール別のソフトウェア開発を行うのは、難しいことではありません。では、例題として、9章の簡易版メモリダンプ・プログラムのソース・プログラムを基に、これを無理(?)に4つのモジュールに分割し、モジュール別ソフトウェア開発法により、完動するダンプ・プログラムを作成してみましょう。

本来は、1つのソース・プログラムになっているものを分割するのではなく、最初からモジュールに分割した開発計画を立て、それぞれのソース・プログラムを独立に作成するわけです。プログラム全体を見通し、それを階層的にいくつかのモジュールに分割して、それぞれの仕様を決定するのは、開発の全体で最も重要でかつ困難な作業でしょう。

では、4つのモジュールに分割したダンプ・プログラムの、各モジュールごとのソース・プログラムを示します。M80によりアセンブルしますので、各ソースファイル名は、「xxxx . MAC」とします。もとのプログラムは、**図9-6-2**および**図9-6-9**のアセンブルリストのものと同じですので、比較してみてください。

図11-2-3　EQU定義のモジュールのソース・プログラム

```
      EQU定義モジュールのソースファイル名
A>TYPE DUEQU.MAC ↵ ………… ファイル「DUEQU.MAC」をタイプアウトする

;--------------------------------------;
;          MEMORY DUMP PROGRAM         ;
;           ( SYMBOL DEFINE )          ;
;--------------------------------------;
        PUBLIC  BDOS
        PUBLIC  PCIN
        PUBLIC  PCOUT  ………… これらのシンボルは、当モジュールで定義しているものであるが、
        PUBLIC  CR              これらを擬似命令「PUBLIC」により、外部のモジュールでも使用
        PUBLIC  LF              可能なシンボルとする
;
BDOS    EQU     0005H
PCIN    EQU     3583H
PCOUT   EQU     3E0DH
CR      EQU     0DH
LF      EQU     0AH
;
        END

A> ………… タイプアウトが終了した後のCP/Mのプロンプト
```

図11-2-4　メイン・モジュールのソース・プログラム

```
      メイン・モジュールのソースファイル名
A>TYPE DUMAIN.MAC ↵

;--------------------------------------;
;          MEMORY DUMP PROGRAM         ;
;           ( MAIN ROUTINE )           ;
;--------------------------------------;
        .Z80 ………… Z-80のニーモニックを使用していることを宣言する擬似命令。前章の図10-3-2のリストの冒頭部参照
        MACLIB  DUIFDEF …… 条件アセンブル設定のマクロ・ライブラリ「DUIFDEF」をここに読み込むことを指定する擬似命令
;
        EXTRN   CR
        EXTRN   STACK
        EXTRN   ADRBUF
        EXTRN   CHRIN  ………… これらのシンボルやラベルは、外部のモジュールで定義されているもの
        EXTRN   CHROUT          であり、これらを擬似命令「EXTRN」により、当モジュールで使用する
        EXTRN   CRLFOUT         ことを可能とする
        EXTRN   HEXOUT
;
        CSEG
START:
        IF      CPM
        LD      SP,STACK
        ENDIF
;
        LD      DE,ADRBUF
        CALL    CRLFOUT
        LD      A,'-'
        CALL    CHROUT
;
        LD      B,4
```

```
NEXIN:
        CALL    CHRIN
        LD      (DE),A
        CP      CR
        JP      Z,AHXBIN
        INC     DE
        DEC     B
        JP      NZ,NEXIN
        LD      A,CR
        LD      (DE),A
;
;------ this routine converts ASCII 2byte
;               digits into BINARY HEX.
;       DE: ASCII data pointer
;       HL: converted data
;
AHXBIN:
        LD      HL,0
        LD      DE,ADRBUF
AHXBI1:
        LD      A,(DE)
        CP      CR
        JP      Z,ADROUT
        ADD     HL,HL
        ADD     HL,HL
        ADD     HL,HL
        ADD     HL,HL
        CALL    HCONV
        JP      NC,INERR
        ADD     A,L
        LD      L,A
        INC     DE
        JP      AHXBI1
HCONV:
        SUB     30H
        CP      0AH
        RET     C
        SUB     7
        CP      10H
        RET
;
;------ input data is not valid hex value
INERR:
        CALL    CRLFOUT
        LD      A,'?'
        CALL    CHROUT
        JP      START
;
;------ address out
ADROUT:
        CALL    CRLFOUT
        LD      A,H
        CALL    HEXOUT
        LD      A,L
        CALL    HEXOUT
        LD      A,':'
        CALL    CHROUT
        LD      A,' '
        CALL    CHROUT
```

```
;
;------ memory data out
        LD      A,(HL)
        CALL    HEXOUT
;
;------ check continue or new address
        CALL    CHRIN
        CP      'E'
        JP      Z,EXIT
        CP      'e'
        JP      Z,EXIT
        CP      CR
        JP      NZ,START
        INC     HL
        JP      ADROUT
;
;------ exit this program
EXIT:
        IF      CPM
        JP      0000
        ENDIF
;
        IF      PC88
        RST     38H
        ENDIF
;
        END

A>
```

図11-2-5 サブルーチン・モジュールのソース・プログラム

サブルーチン・モジュールのソースファイル名

```
A>TYPE DUSUB.MAC

;------------------------------------;
;       MEMORY DUMP PROGRAM          ;
;         ( SUB ROUTINE )            ;
;------------------------------------;
        .Z80
        MACLIB  DUIFDEF
;
        EXTRN   BDOS
        EXTRN   CR
        EXTRN   LF
        PUBLIC  CHRIN
        PUBLIC  CHROUT
        PUBLIC  CRLFOUT
        PUBLIC  HEXOUT
        PUBLIC  ADRBUF
        PUBLIC  STACK
;
        CSEG
;
;------ this sub routine display 1byte binary
;           data as HEX 8 bit value.
;         A: BINARY data --> display as HEX
```

（.Z80, MACLIB DUIFDEF）メイン・モジュールと同じ使い方

（EXTRN BDOS, CR, LF）外部モジュールのこれらのシンボルを当モジュールで使用する

（PUBLIC CHRIN〜STACK）当モジュールでのサブルーチンなどのラベルであるが，これらを外部モジュールでも使用可能とする

```
HEXOUT:
        PUSH    AF
        RRCA
        RRCA
        RRCA
        RRCA
        CALL    HEXOU1
        POP     AF
HEXOU1:
        AND     0FH
        ADD     A,30H
        CP      3AH
        JP      C,HEXOU2
        ADD     A,7
HEXOU2:
        CALL    CHROUT
        RET
;
;------ 1 character out subroutine
CHROUT:
        IF      CPM
        PUSH    DE
        PUSH    HL
        LD      C,2
        LD      E,A
        CALL    BDOS
        POP     HL
        POP     DE
        RET
        ENDIF
;
        IF      PC88
        PUSH    DE
        PUSH    HL
        CALL    PCOUT
        POP     HL
        POP     DE
        RET
        ENDIF
;
;------ carriage return / line feed out subr.
CRLFOUT:
        LD      A,CR
        CALL    CHROUT
        LD      A,LF
        CALL    CHROUT
        RET
;
;------ 1 character key input subroutine
CHRIN:
        IF      CPM
        PUSH    BC
        PUSH    DE
        PUSH    HL
        LD      C,1
        CALL    BDOS
        POP     HL
        POP     DE
        POP     BC
```

```
         RET
         ENDIF
;
         IF       PC88
         PUSH     BC
         PUSH     DE
         PUSH     HL
         CALL     PCIN
         CALL     PCOUT
         POP      HL
         POP      DE
         POP      BC
         RET
         ENDIF
;
;------ input buffer and stack area
ADRBUF   EQU      $
         DS       5
;
         IF       CPM
         DS       32
STACK    EQU      $
         ENDIF
;
         END

A>
```

図11-2-6　条件アセンブル設定マクロ・ライブラリ

条件アセンブルの条件設定マクロ・ライブラリのファイル名

```
A>TYPE DUIFDEF.MAC ↵

;------ if-endif definition macro lib.
TRUE     EQU      0FFFFH
FALSE    EQU      NOT TRUE
;
CPM      EQU      TRUE
PC88     EQU      NOT CPM
;------ end maclib

A>
```

条件アセンブルの条件設定部．他のモジュールに組み込まれるライブラリとして使用する

このモジュールには「END」擬似命令を書いてはいけない(このモジュールは，このままでアンンブルされるわけではなく，そっくりそのまま外部のモジュールに組み込まれるため)

最後に示したモジュールは，その他のモジュールとは使い方が異なり，「マクロ・ライブラリ」として使われます．マクロ・ライブラリのファイルは，それ自身をその場でアセンブルして，オブジェクト・プログラムを作り出すのではなく，他のモジュールがアセンブルされる際に，その中に「MACLIB」擬似命令の指定があれば，指定されたマクロ・ライブラリがソース・プログ

ラムの形で取り込まれ，その中で一緒にアセンブルされるものです．後ほどその具体例を示します.*

では次に，条件アセンブル設定部のマクロ・ライブラリを除いて，3つのモジュールのソース・プログラムを，M80でアセンブルします．それにより3種類のリロケータブル・オブジェクト・プログラムが生成されるまでの実行例を示します．

**図11-2-7　各モジュールのソース・プログラムのアセンブル**

```
A>DIR DU*.MAC ↵ ………実行前のすべてのソースファイルを確認

A: DUEQU      MAC : DUIFDEF   MAC : DUMAIN     MAC : DUSUB      MAC
   シンボル定義モジュール    条件アセンブル        メイン・モジュール    サブルーチン・モジュール
                            指定モジュール
A>M80 DUEQU,DUEQU=DUEQU ↵ ………まず最初にシンボル定義モジュールをアセンブルする
                              (どのモジュールからアセンブルしてもよい)
No Fatal error(s)
アセンブル終了
A>DIR DUEQU.* ↵ ………シンボル定義モジュールに関するファイルの確認

A: DUEQU      MAC : DUEQU     REL : DUEQU     PRN
                    生成されたリロケータブル・  生成されたアセンブルリスト
                    オブジェクト・プログラム
A>M80 DUMAIN,DUMAIN=DUMAIN ↵ ………次にメイン・モジュールをアセンブル

No Fatal error(s)
アセンブル終了
A>DIR DUMAIN.* ↵ ………メイン・モジュールに関するファイルの確認

A: DUMAIN     MAC : DUMAIN    REL : DUMAIN    PRN
                    生成されたリロケータブル・  生成されたアセンブルリスト
                    オブジェクト・プログラム
A>M80 DUSUB,DUSUB=DUSUB ↵ ………次にサブルーチン・モジュールをアセンブルする

No Fatal error(s)
アセンブル終了
A>DIR DUSUB.* ↵ ………サブルーチン・モジュールに関するファイルの確認

A: DUSUB      MAC : DUSUB     REL : DUSUB     PRN
                    生成されたリロケータブル・  生成されたアセンブルリスト
                    オブジェクト・プログラム
A>DIR DU*.REL ↵ ………以上のアセンブル作業で生成されたすべてのリロケータブル・オブジェクトのファイルを確認

A: DUEQU      REL : DUMAIN    REL : DUSUB     REL

A>            この3つのリロケータブル・オブジェクト・プログラム
              がリンクローダでリンクされる
```

*これは，マクロアセンブラのマクロ機能の中で，最も単純なもののひとつである．

生成された3つのモジュールのリロケータブル・オブジェクト・プログラムは，ソース・プログラムのロケーション・カウンタの設定を，いずれも「CSEG」で行っていますので，任意のアドレスにリンクロード可能です．もし，絶対ロード・アドレスを持ったオブジェクト・プログラムを生成する場合は，9章の図**9－6－2**のリストのように，「ASEG」および「ORG」擬似命令を使います．

では，この3つのリロケータブル・オブジェクト・プログラムをL80でリンクし，CP/M上で実行できるプログラムとするために，$0100_H$スタートの実行可能なオブジェクト・プログラムを作成しましょう．その実行例を次に示します．

**図11-2-8　3つのリロケータブル・オブジェクト・プログラムをL80でリンクする**

リンクローダ「L80」を実行し，
スタート・アドレス(ロード・アドレス)を0100Hとして，
この3つのリロケータブル・オブジェクトをリンクし，
インテルHEX形式のオブジェクトにして，ファイル名を「DUMP100」としてディスクにセーブせよ，という意味

```
A>L80 /P:100,DUEQU,DUMAIN,DUSUB,DUMP100/N/X/E

Link-80  3.44  09-Dec-81  Copyright (c) 1981 Microsoft

Data    0100    01E3    <  227>

40647 Bytes Free
[0000   01E3       1]
```

（Data行の注）1本のソース・プログラムとしてアセンブルおよびリンクした9章の図9-6-1のリストと同じ結果である

リンクロード終了

```
A>DIR DUMP100.*
```

リンクローダの実行により生成されたファイルの確認

```
A: DUMP100  HEX
```

3つのモジュールのリロケータブル・オブジェクトが結合され，1本のHEX形式のオブジェクトとして生成されている

```
A>TYPE DUMP100.HEX
```

それをタイプアウトして確認

```
:2001000031E30111BE01CDA7013E2DCD9C010604CDB20112FE0DCA21011305C210013E0DE7
:200120001221000011BE011AFE0DCA520129292929CD3D01D24701856F13C32701D630FEBB
:200140000AD8D607FE10C9CDA7013E3FCD9C01C30001CDA7017CCD84017DCD84013E3ACD92
:200160009C013E20CD9C017ECD8401CDB201FE45CA8101FE65CA8101FE0DC2000123C35286
:2001800001C30000F50F0F0F0FCD8D01F1E60FC630FE3ADA9801C607CD9C01C9D5E50E02BE
:2001A0005FCD0500E1D1C93E0DCD9C013E0ACD9C01C9C5D5E50E01CD0500E1D1C1C9C359AB
:2001C0002B3EFF32F23CCDC428AF32F23C7AB7C29D043A113EFE20C07B3DF8CA9D04FE106B
:0301E000D29D04A9
:00000001FF
```

ロード・アドレス

5バイトのダンプ・アドレス・バッファと32バイトのスタックエリア計37バイトのエリアが確保されているだけで，ここのデータは意味がなく不定である

オブジェクト・プログラム

これで，$0100_H$をスタート・アドレスとするダンプ・プログラムの，インテル HEX 形式オブジェクト・プログラムができあがりました（インテル HEX 形式のオブジェクトを生成することなく，直接「．COM」ファイルを生成することもできる）．これを CP/M の「LOAD」プログラムにより，実行可能な「．COM」形式に変換して，ダンプ・プログラムを実行してみましょう．その実行例を次に示します．

**図11-2-9 できあがったダンプ・プログラムの実行**

```
A>LOAD DUMP100↵ ………CP/Mのローダ「LOAD」により，HEX形式・実行可能形式への変換を行う

FIRST ADDRESS 0100
LAST  ADDRESS 01E2
BYTES READ    00E3
RECORDS WRITTEN 02


A>DIR DUMP100.*↵ ………結果の確認

A: DUMP100  HEX : DUMP100  COM
                  生成された実行可能なオブジェクト・プログラム

A>DUMP100↵ ………できあがったダンプ・プログラムの実行
ダンプ・プログラムが起動した
-FE↵ ………アドレス00FEHをダンプする
00FE: E5↵
00FF: E5↵
0100: 31↵
0101: E3↵  リターンキーの入力により，次々とダンプしていく
0102: 01↵
0103: 11↵
0104: BE↵
0105: 01x ………リターンキー以外を入力すると，新しいダンプ・アドレスの入力が可能となる
-1BD↵ ………アドレス01BDHをダンプ，01BEHからは当プログラムのダンプ・アドレス・バッファである．
01BD: C9↵
01BE: 31↵
01BF: 42↵  31H＝「1」，42H＝「B」，44H＝「D」，と格納されている
01C0: 44↵
01C1: 0D↵
01C2: FFx
-FFFE
FFFE: FF↵
FFFF: 00↵  FFFFFHの次は0000Hとなる
0000: C3↵
0001: 03↵
0002: DA↵
0003: 81e ………eの入力で当プログラムを終了する

A> ………CP/Mに戻った
```

モジュール別ソフトウェア開発法で作成したこのダンプ・プログラムの実行可能なオブジェクト・プログラムは，９章で作成したものとまったく同一であることがわかります．比較してみてください．

では次に，アセンブル時に生成された３つのモジュールのアセンブルリストを示しましょう．このアセンブルリストには，リロケータブル・マクロアセンブラのいろいろな機能の結果が表れていますので，それをリスト上で解説します．

図11-2-10　EQU定義モジュールのアセンブルリスト

```
        MACRO-80 3.44   09-Dec-81       PAGE    1

                                ;---------------------------------------;
                                ;          MEMORY DUMP PROGRAM          ;
                                ;           ( SYMBOL DEFINE )           ;
                                ;---------------------------------------;
                                        PUBLIC  BDOS
                                        PUBLIC  PCIN
                                        PUBLIC  PCOUT
                                        PUBLIC  CR
                                        PUBLIC  LF
                                ;
0005                            BDOS    EQU     0005H
3583                            PCIN    EQU     3583H
3E0D                            PCOUT   EQU     3E0DH
000D                            CR      EQU     0DH
000A                            LF      EQU     0AH
                                ;
                                        END
```

このモジュールは，EQUなどの擬似命令だけの内容なので，
オブジェクト・プログラムは生成されない

図11-2-11　メイン・モジュールのアセンブルリスト

```
        MACRO-80 3.44   09-Dec-81       PAGE    1

                                ;---------------------------------------;
                                ;          MEMORY DUMP PROGRAM          ;
                                ;           ( MAIN ROUTINE )            ;
                                ;---------------------------------------;
                                        .Z80
                        C               MACLIB  DUIFDEF
                        C       ;------ if-endif definition macro lib.
FFFF                    C       TRUE    EQU     0FFFFH
0000                    C       FALSE   EQU     NOT TRUE
                        C       ;
FFFF                    C       CPM     EQU     TRUE
0000                    C       PC88    EQU     NOT CPM
                        C       ;------ end maclib
                                ;
```

外部のファイル「DUIFDEF」をこの位置に読み込む擬似命令「MACLIB」

この「C」記号は，この間のプログラムが，外部のライブラリファイルから読み込まれたものであることを示している

条件アセンブルは「CPM」側をアセンブルする

「'」記号は，このアドレスが絶対アドレスではなく，相対的なアドレスであることを示している

「*」記号は，外部モジュールのシンボルやラベルであることを示す．アセンブル時にはその値はまだ入力されず，00が代入されている．これらの実の値は，リンクロード時に与えられる

```
                                  EXTRN   CR
                                  EXTRN   STACK
                                  EXTRN   ADRBUF
                                  EXTRN   CHRIN
                                  EXTRN   CHROUT
                                  EXTRN   CRLFOUT
                                  EXTRN   HEXOUT
                         ;
0000'                             CSEG
0000'                    START:
                                  IF      CPM
0000'    31 0000*                 LD      SP,STACK
                                  ENDIF
                         ;
0003'    11 0000*                 LD      DE,ADRBUF
0006'    CD 0000*                 CALL    CRLFOUT
0009'    3E 2D                    LD      A,'-'
000B'    CD 0000*                 CALL    CHROUT
                         ;
000E'    06 04                    LD      B,4
0010'                    NEXIN:
0010'    CD 0000*                 CALL    CHRIN
0013'    12                       LD      (DE),A
0014'    FE 00*                   CP      CR
0016'    CA 0021'                 JP      Z,AHXBIN
0019'    13                       INC     DE
001A'    05                       DEC     B
001B'    C2 0010'                 JP      NZ,NEXIN
001E'    3E 00*                   LD      A,CR
0020'    12                       LD      (DE),A
                         ;
                         ;------ this routine converts ASCII 2byte
                         ;                  digits into BINARY HEX.
                         ;      DE: ASCII data pointer
                         ;      HL: converted data
                         ;
0021'                    AHXBIN:
0021'    21 0000                  LD      HL,0
0024'    11 0000*                 LD      DE,ADRBUF
0027'                    AHXBI1:
0027'    1A                       LD      A,(DE)
0028'    FE 00*                   CP      CR
002A'    CA 0052'                 JP      Z,ADROUT
002D'    29                       ADD     HL,HL
002E'    29                       ADD     HL,HL
002F'    29                       ADD     HL,HL
0030'    29                       ADD     HL,HL
0031'    CD 003D'                 CALL    HCONV
0034'    D2 0047'                 JP      NC,INERR
0037'    85                       ADD     A,L
0038'    6F                       LD      L,A
0039'    13                       INC     DE
003A'    C3 0027'                 JP      AHXBI1
003D'                    HCONV:
003D'    D6 30                    SUB     30H
003F'    FE 0A                    CP      0AH
0041'    D8                       RET     C
0042'    D6 07                    SUB     7
0044'    FE 10                    CP      10H
0046'    C9                       RET
```

```
                                        ;
                                        ;------ input data is not valid hex value
0047'                                   INERR:
0047'    CD 0000*                                CALL    CRLFOUT
004A'    3E 3F                                   LD      A,'?'
004C'    CD 0000*                                CALL    CHROUT
004F'    C3 0000'                                JP      START
                                        ;
                                        ;------ address out
0052'                                   ADROUT:
0052'    CD 0000*                                CALL    CRLFOUT
0055'    7C                                      LD      A,H
0056'    CD 0000*                                CALL    HEXOUT
0059'    7D                                      LD      A,L
005A'    CD 0000*                                CALL    HEXOUT
005D'    3E 3A                                   LD      A,':'
005F'    CD 0000*                                CALL    CHROUT
0062'    3E 20                                   LD      A,' '
0064'    CD 0000*                                CALL    CHROUT
                                        ;
                                        ;------ memory data out
0067'    7E                                      LD      A,(HL)
0068'    CD 0000*                                CALL    HEXOUT
                                        ;
                                        ;------ check continue or new address
006B'    CD 0000*                                CALL    CHRIN
006E'    FE 45                                   CP      'E'
0070'    CA 0081'                                JP      Z,EXIT
0073'    FE 65                                   CP      'e'
0075'    CA 0081'                                JP      Z,EXIT
0078'    FE 00*                                  CP      CR
007A'    C2 0000'                                JP      NZ,START
007D'    23                                      INC     HL
007E'    C3 0052'                                JP      ADROUT
                                        ;
                                        ;------ exit this program
0081'                                   EXIT:
                                                 IF      CPM
0081'    C3 0000                                 JP      0000
                                                 ENDIF
                                        ;
                                                 IF      PC88
                                                 RST     38H
                                                 ENDIF
                                        ;
                                                 END
```

**図11-2-12　サブルーチン・モジュールのアセンブルリスト**

```
      MACRO-80 3.44   09-Dec-81        PAGE    1

                                   ;---------------------------------------;
                                   ;          MEMORY DUMP PROGRAM          ;
                                   ;           ( SUB ROUTINE )             ;
                                   ;---------------------------------------;
                                           .Z80
                          C                MACLIB  DUIFDEF
                          C        ;------ if-endif definition macro lib.
FFFF                      C        TRUE    EQU     0FFFFH
0000   メイン・モジュール C        FALSE   EQU     NOT TRUE
       の場合と同じ       C        ;
FFFF                      C        CPM     EQU     TRUE
0000                      C        PC88    EQU     NOT CPM
                          C        ;------ end maclib
                                   ;
                                           EXTRN   BDOS
                                           EXTRN   CR
                                           EXTRN   LF
                                           PUBLIC  CHRIN
                                           PUBLIC  CHROUT
                                           PUBLIC  CRLFOUT
                                           PUBLIC  HEXOUT
                                           PUBLIC  ADRBUF
                                           PUBLIC  STACK
                                   ;
0000'                                      CSEG
                                   ;
                                   ;------ this subroutine display 1byte binary
                                   ;            data as HEX 8 bit value.
                                   ;        A: BINARY data --> display as HEX
0000'                              HEXOUT:
0000'   F5                                 PUSH    AF
0001'   0F                                 RRCA
0002'   0F                                 RRCA
0003'   0F                                 RRCA
0004'   0F                                 RRCA
0005'   CD 0009'                           CALL    HEXOU1
0008'   F1                                 POP     AF
0009'                              HEXOU1:
0009'   E6 0F                              AND     0FH
000B'   C6 30                              ADD     A,30H
000D'   FE 3A                              CP      3AH
000F'   DA 0014'                           JP      C,HEXOU2
0012'   C6 07                              ADD     A,7
0014'                              HEXOU2:
0014'   CD 0018'                           CALL    CHROUT
0017'   C9                                 RET
                                   ;
                                   ;------ 1 character out subroutine
0018'                              CHROUT:
                                           IF      CPM
0018'   D5                                 PUSH    DE
0019'   E5                                 PUSH    HL
001A'   0E 02                              LD      C,2
001C'   5F                                 LD      E,A
001D'   CD 0000*                           CALL    BDOS
0020'   E1                                 POP     HL
```

```
0021'   D1                          POP     DE
0022'   C9                          RET
                                    ENDIF
                        ;
                                    IF      PC88
                                    PUSH    DE
                                    PUSH    HL
                                    CALL    PCOUT
                                    POP     HL
                                    POP     DE
                                    RET
                                    ENDIF
                        ;
                        ;------ carriage return / line feed out subr.
0023'                   CRLFOUT:
0023'   3E 00*                      LD      A,CR
0025'   CD 0018'                    CALL    CHROUT
0028'   3E 00*                      LD      A,LF
002A'   CD 0018'                    CALL    CHROUT
002D'   C9                          RET
                        ;
                        ;------ 1 character key input subroutine
002E'                   CHRIN:
                                    IF      CPM
002E'   C5                          PUSH    BC
002F'   D5                          PUSH    DE
0030'   E5                          PUSH    HL
0031'   0E 01                       LD      C,1
0033'   CD 0000*                    CALL    BDOS
0036'   E1                          POP     HL
0037'   D1                          POP     DE
0038'   C1                          POP     BC
0039'   C9                          RET
                                    ENDIF
                        ;
                                    IF      PC88
                                    PUSH    BC
                                    PUSH    DE
                                    PUSH    HL
                                    CALL    PCIN
                                    CALL    PCOUT
                                    POP     HL
                                    POP     DE
                                    POP     BC
                                    RET
                                    ENDIF
                        ;
                        ;------ input buffer and stack area
003A'                   ADRBUF      EQU     $
003A'                               DS      5
                        ;
                                    IF      CPM
003F'                               DS      32
005F'                   STACK       EQU     $
                                    ENDIF
                        ;
                                    END
```

現在，ディスク上には，ダンプ・プログラムを構成する3つのモジュールのリロケータブル・オブジェクト・プログラム(「DUEQU . REL」,「DUMAIN . REL」,「DUSUB . REL」)があります．これらはいずれも「CSEG」指定によるオブジェクト・プログラムですから，メモリ上にロードする際のアドレス情報は何も持っていません．よって，この3つのリロケータブルなオブジェクト・プログラムは，リンクローダにより，任意のスタート・アドレス(ロード・アドレス)を持つ実行可能なオブジェクト・プログラムを生成することができます．

先ほどは，$0100_H$スタートのダンプ・プログラムを，L80のリンク作業によって作成しましたが，ここでは，同じ3つのリロケータブル・オブジェクトから，$4000_H$スタートのオブジェクト・プログラムを作成してみましょう．メイン・モジュールのロードアドレスを$4000_H$に指定して，L80によるリンク作業を行えばよいわけです．ではその実行例を次に示します．

**図11-2-13　$4000_H$スタートのオブジェクトを作成するL80のリンク作業**

```
A>DIR DU*.REL ↵ ……………すべてのリロケータブル・オブジェクト・ファイルを確認

A: DUEQU     REL : DUMAIN    REL : DUSUB     REL
                  この3つをリンクする
A>DIR DUIFDEF.MAC ↵ ………ライブラリファイルを確認(存在する必要はないが)

A: DUIFDEF  MAC ……………このファイルは，アセンブル時に必要なもので，リンクローダの実行の際には必要なし

      今回のロード・アドレスは4000H         今回のリンク後のプログラムファイル名は「DUMP4000」とする
A>L80 /P:4000,DUEQU,DUMAIN,DUSUB,DUMP4000/N/X/E ↵ ……リンクローダの実行．図11-2-8の
                                                        実行例と同じ
Link-80  3.44  09-Dec-81  Copyright (c) 1981 Microsoft

Data    4000    40E3    <  227>

40647 Bytes Free
[0000   40E3       64]


A>DIR DUMP4000.* ↵ ………生成されたHEX形式のオブジェクト・プログラムの確認

A: DUMP4000 HEX
   生成されたHEX形式の
A> オブジェクト・プログラム
```

この作業で，$4000_H$スタートのインテルHEX形式のオブジェクト・プログラムができました．これをCP/MのデバッガDDTにより$4000_H$からのメモリにロードして，このダンプ・プログラムを実行してみましょう．

**図11-2-14 4000$_H$スタートのダンプ・プログラムの実行**

```
A>DDT DUMP4000.HEX↵ ………DDTを起動して,HEX形式のダンプ・プログラムをメモリにロードする

DDT VERS 2.2
NEXT  PC
40E3 0000          DDTが起動して,HEX形式のオブジェクト自身が持っているロード・アドレスに,
                   実行可能なオブジェクトに変換してロードされた

-D3FF0 40FF↵ ………4000H からロードされたダンプ・プログラムをDDTでダンプ
3FF0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
4000 31 E3 40 11 BE 40 CD A7 40 3E 2D CD 9C 40 06 04 1.@..@..@>-..@..
4010 CD B2 40 12 FE 0D CA 21 40 13 05 C2 10 40 3E 0D ..@....!@....@>.
4020 12 21 00 00 11 BE 40 1A FE 0D CA 52 40 29 29 29 .!....@....R@)))
4030 29 CD 3D 40 D2 47 40 85 6F 13 C3 27 40 D6 30 FE ).=@.G@.o..'@.0.
4040 0A D8 D6 07 FE 10 C9 CD A7 40 3E 3F CD 9C 40 C3 .........@>?..@.
4050 00 40 CD A7 40 7C CD 84 40 7D CD 84 40 3E 3A CD .@..@|..@}..@>:.
4060 9C 40 3E 20 CD 9C 40 7E CD 84 40 CD B2 40 FE 45 .@> ..@~..@..@.E
4070 CA 81 40 FE 65 CA 81 40 FE 0D C2 00 40 23 C3 52 ..@.e..@....@#.R
4080 40 C3 00 00 F5 0F 0F 0F 0F CD 8D 40 F1 E6 0F C6 @..........@....
4090 30 FE 3A DA 98 40 C6 07 CD 9C 40 C9 D5 E5 0E 02 0.:..@....@.....
40A0 5F CD 05 00 E1 D1 C9 3E 0D CD 9C 40 3E 0A CD 9C _......>...@>...
40B0 40 C9 C5 D5 E5 0E 01 CD 05 00 E1 D1 C1 C9 00 00 @...............
40C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
40D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
40E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
40F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................

-G4000↵ ………アドレス4000H から実行      5バイトのダンプ・アドレス・バッファ
ダンプ・プログラムが起動した              と32バイトのスタックエリア
-4000………プログラムがロードされるアドレスが異なっていても,
4000: 31↵  ダンプ・プログラムは同様に実行される           オブジェクト・プログラム
4001: E3↵  (4000H 付近のダンプデータを，上のDDTによるもの
4002: 40↵  と比較すると，当り前だが一致する)
4003: 11↵
4004: BEx
-40B0
40B0: 40↵
40B1: C9↵
40B2: C5↵
40B3: D5↵
40B4: E5x
-0↵
0000: C3↵
0001: 03↵
0002: DA↵
0003: 81↵
0004: 00e ………当プログラムを終了する

A>………CP/Mに戻った
```

# 12

# アセンブラから高級言語へ

今までアセンブラの基礎的なことについて，いろいろと解説してきましたが，アセンブラはCPUを直接操作できる唯一の言語であることと，その基本的な使い方は十分に理解できたことと思います．

ところでアセンブラは，CPUのすべての機能を操作できる言語ですが，開発に要する時間的な効率はよくありません．このことは，2章でも述べたように，BASIC言語で，

PRINT "Good Morning"

という，たった1行のステートメントに相当するプログラムを，アセンブラで書くと，図2-1-4のように何行ものプログラムになってしまうことからもわかるでしょう．なにしろアセンブラは，CPUの最も細かい命令語を使ってプログラミングするわけですから，当然，ソース・プログラムのステップ数は，高級言語よりも多くなってしまいます．

アセンブラには，他のどのような言語よりも，最小のサイズ，かつ最高速のオブジェクト・プログラムを作り出すことができるという特長があります．しかし，オブジェクト・プログラムの大きさや，実行スピードの問題等の走行環境が許せば，目的に合った高級言語を使った方が開発作業の能率はずっとよくなるでしょう．このようなことから，規模の大きなプログラムを開発するには，開発効率やプログラムの保守などの問題で，事情が許せば高級言語を使うことになるでしょう．

高級言語には，大きく分けて3つの形態がありますが，一般的な実用ソフトウェアのプログラミングに，アセンブラの代わりとして使用される言語は，コンパイラ言語です．

本書のしめくくりとして，このコンパイラ言語を中心に，その概要を解説しましょう．

# 12-1 コンピュータ言語の種類

コンピュータ言語の種類には，本書でその基礎を学んだアセンブリ言語のほかに，いわゆる「高級言語」と呼ばれる多くの言語があります．その形態を大きく分類すると，

- **コンパイラ言語**
- **インタープリタ言語**
- **中間コード形言語**

の3つの種類があります．それぞれを簡単に解説しましょう．

## コンパイラ言語とインタープリタ言語

マイクロコンピュータ上の代表的なコンパイラ(compiler：辞書での一般的な意味は「編集者」)の一例には，次のような製品があります．

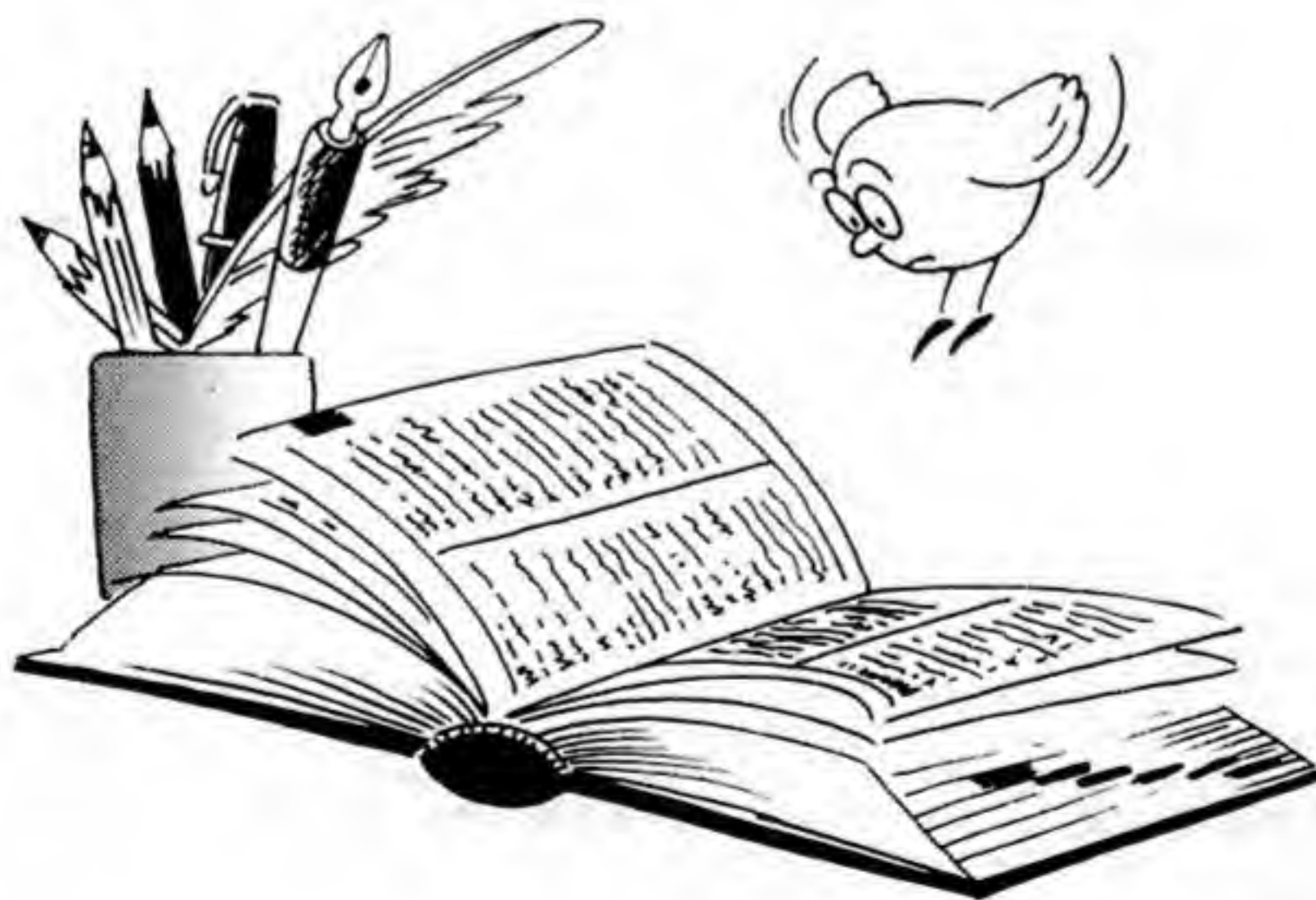
| 製品名 | メーカー | 言語 |
|---|---|---|
| BASIC コンパイラ | マイクロソフト社 | BASIC |
| CB-80 | デジタルリサーチ社 | BASIC |
| FORTRN-80 | マイクロソフト社 | FORTRAN |
| BDS-C コンパイラ | BD Software 社 | C |
| LSI-C コンパイラ | LSI社 | C |
| Pascal／mt＋ | デジタルリサーチ社 | PASCAL |
| TURBO PASCAL | ボーランド・インターナショナル社 | PASCAL |
| PLMX | シスコン社 | PL/M |
| Rgy FORTH | リギーコーポレーション | FORTH |

**図12-1-1 各種のコンパイラ製品**

これらはいずれも，8ビットのCP／M上で実行できる製品であり，ソフトウェア・ショップで入手できます．

コンパイラは，それぞれの言語のソース・プログラムから，オブジェクト・プログラムを作り出します．このことを，BASICインタープリタと，BASICコンパイラとを比較して解説しましょう．

### ＊BASICインタープリタの場合

BASICインタープリタは，多くのパーソナル・コンピュータに付属しているもので，オブジェクト・プログラムは作り出しません．実行時にBASICのソース・プログラムの1行1行を，そのつど解釈し，あらかじめ用意されている実行のためのいろいろなマシン語のルーチンを呼び出しては実行していきます．従って，実行時には，BASICインタープリタ自身が，ソース・プログラムとともにメモリ上に存在していなければ実行することはできません．

## ＊ BASIC コンパイラの場合

BASIC コンパイラは，BASIC 言語で書かれたソース・プログラム（BASIC インタープリタの場合と同じもの）から，コンパイルやリンクロードの作業により，実行可能な純マシンコードのオブジェクト・プログラムを作り出します．これは，アセンブリ・ソース・プログラムからアセンブルとリンクロードの作業により，実行可能な純マシンコードのオブジェクト・プログラムが作り出されるのと同じ形態です．

実行可能な純マシンコードのオブジェクト・プログラムができあがれば，アセンブラから作られたものであろうと，BASIC コンパイラから作られたものであろうと，同じ形態であり，同様に実行できます．よって実行時は，このオブジェクト・プログラムを単独で実行できます．

次に，インタープリタとコンパイラの形態の違いを対比して示しましょう．

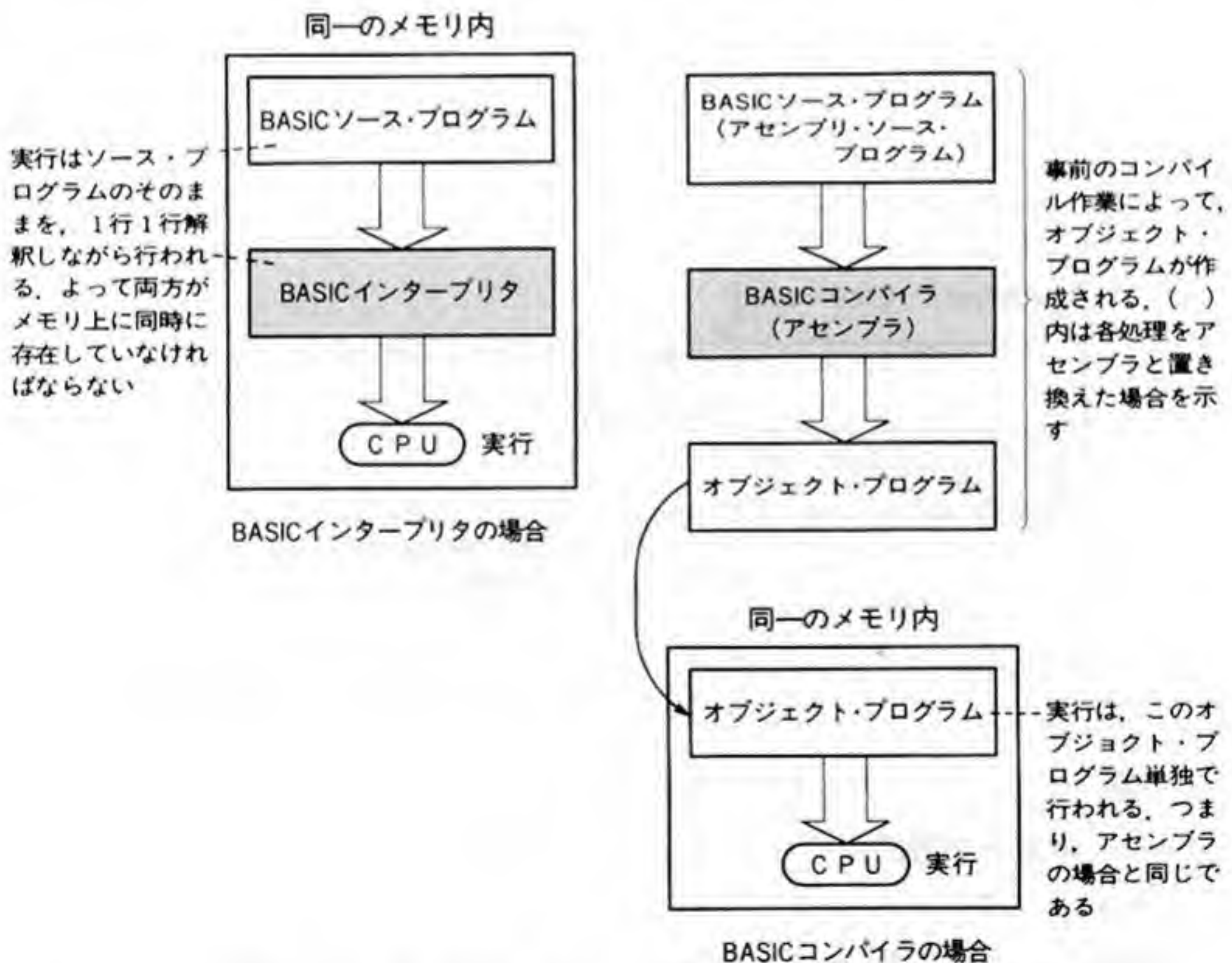



図12-1-2　インタープリタとコンパイラの形態の違い

## 中間コード形言語

中間コード形言語の代表的な例としては,「UCSD Pascal」が挙げられます.これには「Pコード」と呼ばれる中間コードが使われています.中間コード形言語の形態は,コンパイラとインタープリタの中間に位置するものです.この処理を次の図で示しましょう.

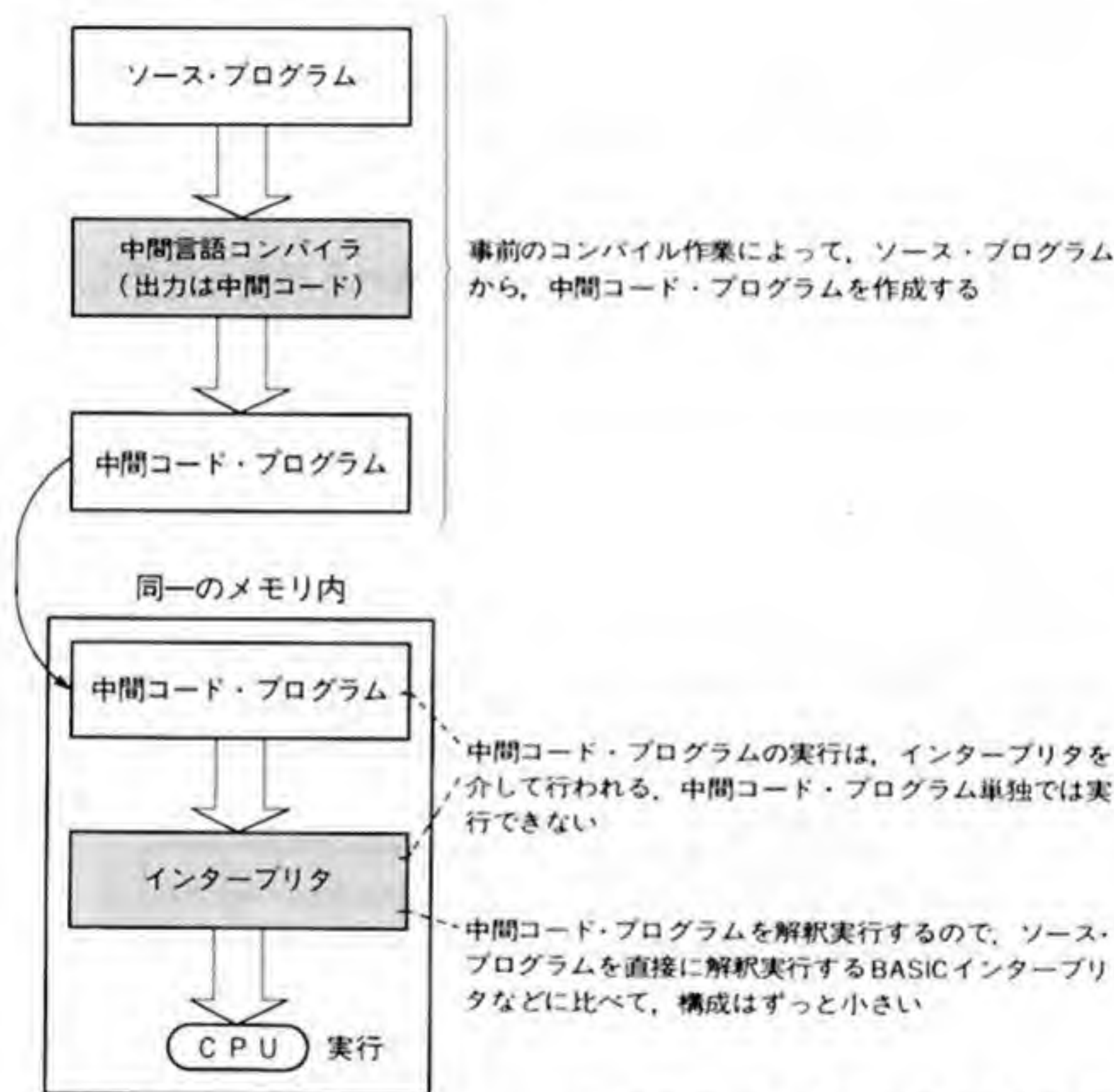



図12-1-3 中間コード形言語の形態

「中間コード」は,ソース・プログラムでも,オブジェクト・プログラムでもない,両者の中間的なプログラムコードであることからこのように呼ばれています.

BASICインタープリタの場合は,ソース・プログラムをそのままインタープリタで解釈して実行しますが,中間コード形の場合は,ソース・プログラムを事前に中間言語コンパイラを使って,中間コードのプログラムに変換し

ておきます．実行時にこれをインタープリタで実行するわけです．よって，事前処理されている分，実行速度はインタープリタよりも速く，インタープリタ部もコンパクトになります．

この中間コード形言語の実行には，インタープリタという余計なものが介在しなくてはならない反面，異種CPU間とのプログラムの互換性については有利になります．つまり，図中の中間コード用のインタープリタ部は，各CPU別にそれぞれを用意しなければならないものですが，ということは，このインタープリタ部を用意しさえすれば，同一の中間コード・プログラムが，各種のCPU上で実行可能になるわけです．

これは，8ビット・マシンのBASICのプログラム(基本的なステートメントでできているもので，アスキーファイルのソース・プログラム)を16ビット・マシン上に持ってきても，そのままで実行可能であることと同じ原理であり，インタープリタが介在することによって生じる当然の機能といえます．*

ここで，アセンブリ言語を含めた4つの言語形態について，オブジェクト・プログラム実行時の相違点などを図解してみましょう．

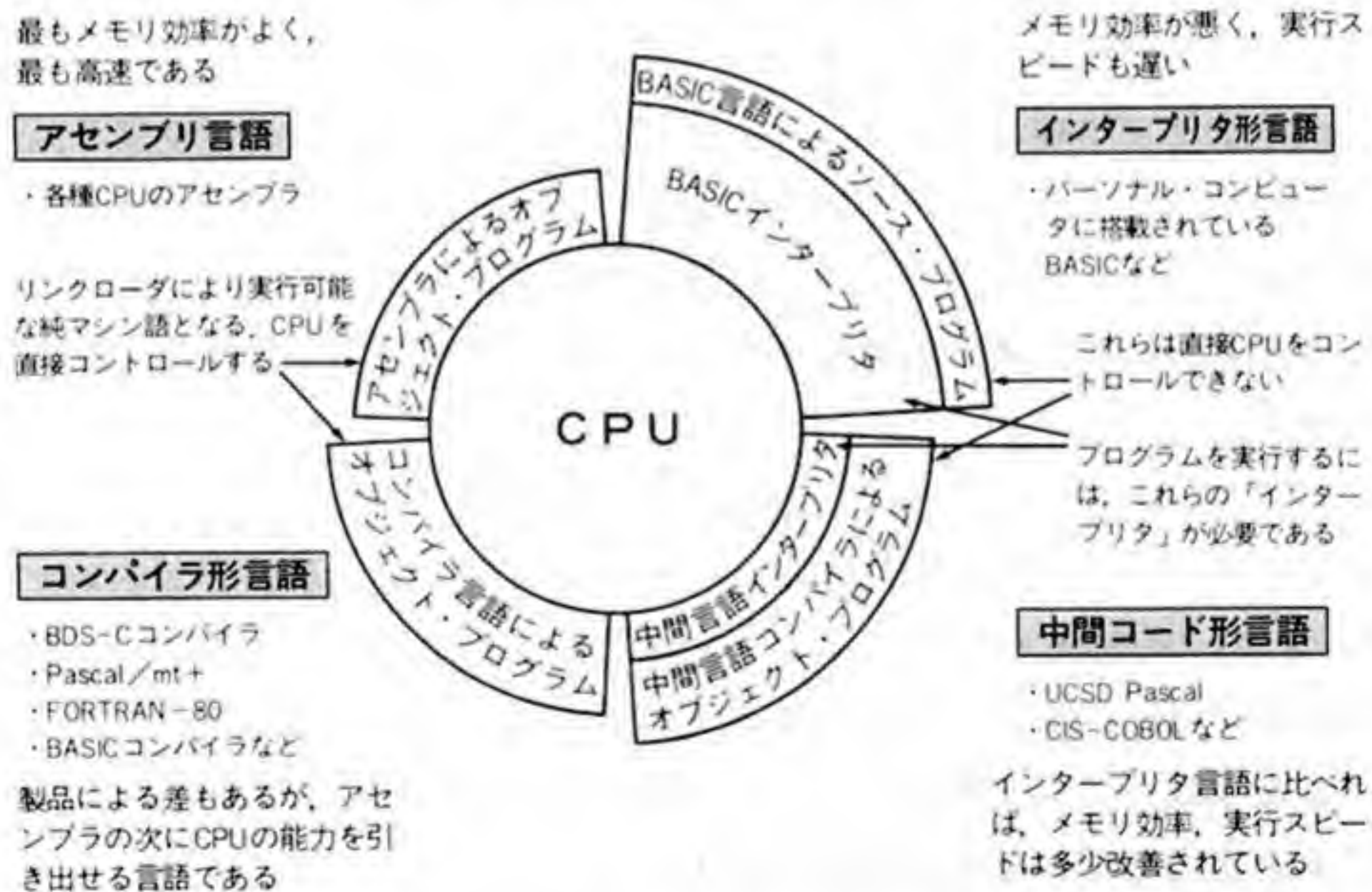



**図12-1-4　4つの言語形態によるプログラム実行時の状態の比較**

＊中間コード形言語の場合，一般にそのコンパイラ部は，その中間コードによって記述される．従って新しいCPUに対しては，中間コード用のインタープリタ部のみを作成すればよい．

# 12 2 BASICコンパイラの実行例

ここでは，BASIC言語のコンパイラを使って，コンパイラによるソフトウェア開発の実例を簡単に紹介します．その前に，コンパイラには大きく分けて2つのタイプがあることを解説しておきましょう．

コンパイラの種類は，それぞれの言語によるソース・プログラムをコンパイルすることにより出力されるプログラムの形態によって，

**(a) オブジェクト・プログラム(多くのものはリロケータブル・オブジェクト・プログラムの形式)を作り出すもの**

**(b) アセンブリ・ソース・プログラムを作り出すもの**

の2つのタイプに分けられます．

(a)のタイプの場合は，リロケータブル・オブジェクト・プログラムが出力されますので，コンパイル後の作業としては，リンクローダを実行し，他のオブジェクト・プログラムや，各種のライブラリ(ルーチン集)中のルーチンなどを結合して，実行可能なオブジェクト・プログラムを作り出すことになります．

このタイプのコンパイラは，それぞれの言語のソース・プログラムからオブジェクト・プログラムを生成する「オブジェクトコード・ジェネレータ」(マシンコード生成ソフト)であるといってもよいでしょう．ただし，このタイプの中には，内部では(b)のタイプのように，アセンブリ・ソース・プログラムを生成して，それをアセンブルすることにより，オブジェクト・プログラムを出力しているものもあります．

(b)のタイプの場合は，アセンブリ・ソース・プログラムが出力されますので，その後の作業はアセンブラの場合と同じです．アセンブラを実行して

オブジェクト・プログラムを生成し，それに対してリンクローダを実行します．必要なら，コンパイラから出力されたアセンブリ・ソース・プログラムにエディタを使って手を入れ，細かい操作や修正を加えることも可能です．

このタイプのコンパイラは，それぞれの言語のソース・プログラムをアセンブリ・ソース・プログラムに変換する「ソースコード・トランスレータ」(ソース・プログラム変換ソフト)であるといってもよいでしょう．

この(a)，(b)の違いを次の図で示します．

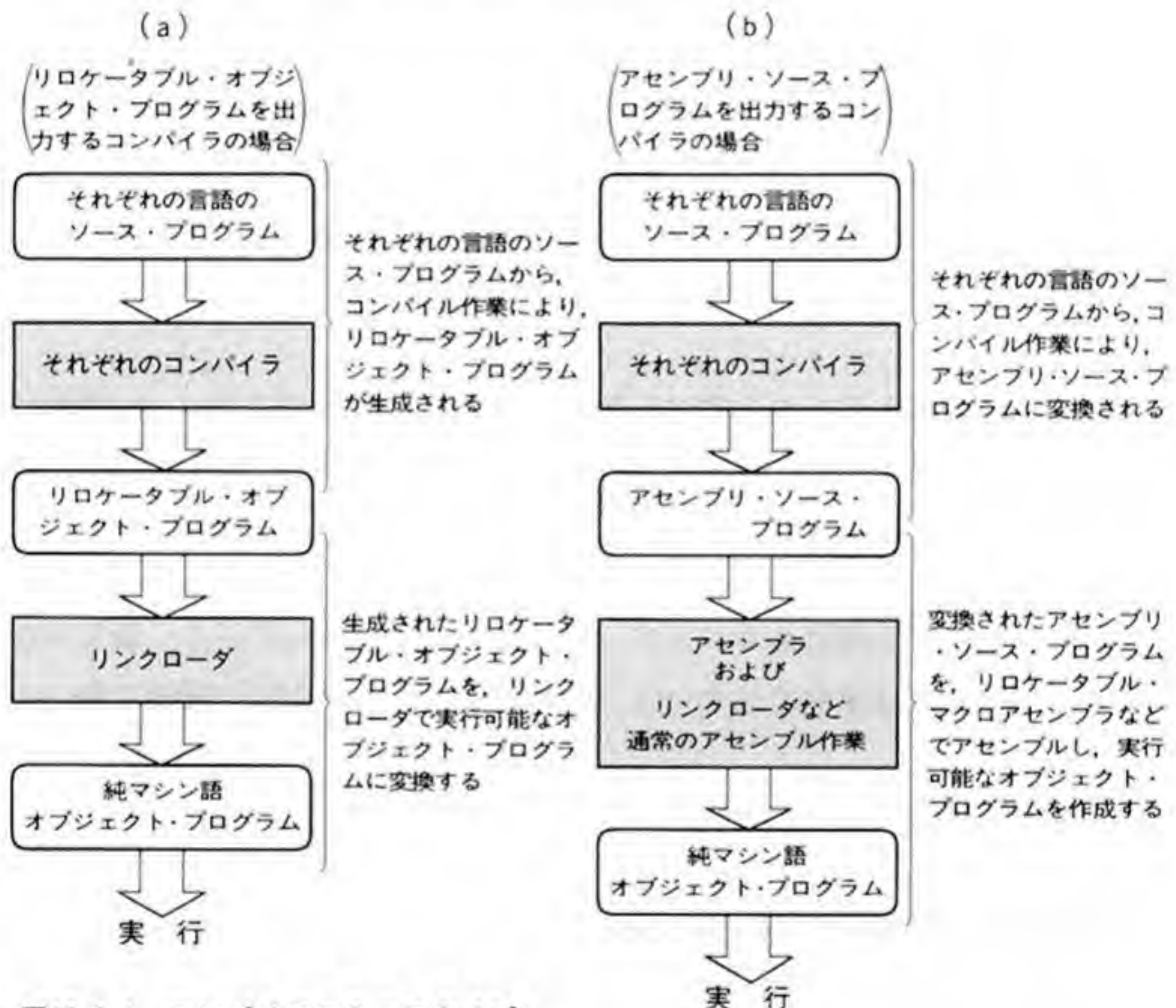



**図12-2-1　コンパイラの2つのタイプ**

では，BASIC言語のコンパイラによるソフトウェア開発の実行例を示しましょう．コンパイラ形のBASICとしては，マイクロソフト社のBASICコンパイラ「BASCOM」が普及していますが，ここでは『はじめて読むマシン語』の序章でも取り上げたデジタルリサーチ社の「CB-80」(米国でビジネス用に広く使われているCBASICのコンパイラ版)を使ってみます．

# BASICコンパイラとインタープリタの実行と比較

では，コンパイラ形のBASIC「CB-80」の実行例を簡単に紹介しましょう．作成するプログラムは，ループを回って単純な計算をするもので，できあがったオブジェクト・プログラムの実行結果(PC-8801上のCP/Mで実行)と，同じPC-8801上の$N_{88}$-BASICによる実行結果とを比較してみます．

作成するプログラムは，1から10000までの間に素数がいくつあるかを求めるものです．この問題のプログラミングには，「エラトステネスのふるい」というアルゴリズムを利用する方法がよく知られていますが，ここでは故意に，非常に能率悪く，しかも実行時間が長くかかる，素朴なアルゴリズムのプログラムを作りました．ソース・プログラムは実行例の中で示しますが，そこで使われている3つの変数と，それらの意味を次に示します．

C———素数かどうかを調べる対象となる数．偶数(2を除く)は素数ではないので，奇数についてのみ(2ずつカウントアップして)調べられる

D———2から順にカウントアップしながらCを割っていく．余りがなければ素数ではないことがわかる．これはMOD関数で調べる

N———素数であると判別されたものの合計の数

まず先に，PC-8801の$N_{88}$-BASICインタープリタで実行した結果を，ソース・プログラム(プログラムファイル名：SOSUN8.BAS)とともに示しましょう．プログラムが起動したときと，終了したときに，時刻を表示するようにしてありますので，所要時間を知ることができます．

このソース・プログラムでは，答が出るまでに25分06秒かかっていますが，結果の「1229」は正しい答です．1から10000までの間の素数は，1229個あります．*

* ラインNo.160のN＝N＋1の後に，PRINT Cを挿入すれば素数そのものを知ることもできる．

図12-2-2 素数の数を求めるプログラムと, $N_{88}$-BASICでの実行例

```
Disk version [Apr 24,1982]
How many files(0-15)?
NEC N-88 BASIC Version 1.0
Copyright (C) 1981 by Microsoft
45530 Bytes free
Ok
LOAD "SOSUN8.BAS"
Ok
LIST
100 DEFINT C,D,N
110 PRINT TIME$
120 C=1 : N=1
130 C=C+2 : D=2
140 IF C>10000 THEN PRINT TIME$ : PRINT N : END
150 IF (C MOD D)=0 THEN GOTO 130 ELSE D=D+1
160 IF D*D > C THEN N=N+1 : GOTO 130 ELSE GOTO 150
Ok
RUN
10:00:21
10:25:27
 1229
Ok
```



では次に，これと同じソース・プログラムをコンパイラ形BASIC「CB-80」でコンパイルし，実行可能な純マシンコードのオブジェクト・プログラムを作成してみましょう．

CB-80には，マイクロソフト系のBASICとは異なった特徴がいくつかありますが，中でもソース・プログラムにラインNo.を必要としないため，行番号の煩わしさから開放されることは特筆すべきことでしょう．ソース・プログラムにおける $N_{88}$-BASICと，CB-80との書式の相違点が数か所ありますが，それを次のページに示します．

**$N_{88}$-BASICの場合**　　**CB-80の場合**

| | | | |
|---|---|---|---|
| • DEFINT | ⇨ | INTEGER | ―――整数宣言 |
| • END | ⇨ | STOP | ―――プログラムの終了 |
| • C　MOD　D | ⇨ | MOD(C , D) | ―――剰余 |

また，CB-80は，時刻を表示するTIME$関数が組み込まれていないので，NECのCP/Mのエスケープ・シーケンスによる時刻表示機能を利用します．エスケープ・シーケンスとは，エスケープコード「$1B_H$」の後に続く文字や文字列をCRTディスプレイに出力することにより，ディスプレイ上の各種のコントロールを行う機能のことです．

時刻を表示するエスケープ・シーケンスは，「ESC　A」です．つまり，$1B_H$，$41_H$と連続して出力することにより，時刻が表示されます．よって，

```
PRINT  TIME$  ⇨  PRINT  CHR$(1BH) ; "A"
```

のように変更します．マイクロソフト系BASICでは，16進数を表すのに「&Hxx」と記述しますが，CB-80では，アセンブラなどで使われている一般的な書き方を用い「xxH」と記述します．

ではCB-80コンパイラを実行しますが，最初に，$N_{88}$-BASICでのプログラムと同じ内容を，CB-80用に変更したソース・プログラム(プログラム名：SOSUCB . BAS)と，CB-80を実行するために必要な主要プログラムファイルをタイプアウトして示します．

コンパイラは，インタープリタのように，ソース・プログラム中のスペースの存在や変数の文字数などによる実行速度の遅れはありません．読みやすいソース・プログラムを自由なレイアウトで作成することができます．

図12-2-3　CB-80の主要プログラムファイルと，例題プログラムのソース

```
A>DIR ↵ ………ディスク上に存在するすべてのファイルのファイル名を表示する．いずれもコンパイルに必要なファイル
   コンパイラのメイン・プログラム                                                       である
A: CB80     COM : CB80     IRL : CB80     OV1 : CB80     OV2
A: CB80     OV3 : LK80     COM : SOSUCB   BAS
                  リンクローダ        ソース・プログラム

A>TYPE SOSUCB.BAS ↵ ……………ソース・プログラムをタイプアウトする

          INTEGER C,D,N
          PRINT CHR$(1BH);"A"
          C=1 : N=1
LOOP1:
          C=C+2 : D=2
            IF C>10000 THEN PRINT CHR$(1BH);"A" : PRINT N : STOP
LOOP2:
              IF (MOD(C,D))=0 THEN GOTO LOOP1 ELSE D=D+1
                IF D*D > C THEN N=N+1 : GOTO LOOP1 ELSE GOTO LOOP2

A>
          ↑
          CB-80用のソース・プログラム．ラインNo.が不要であり，アセンブラに似たラベルの使い方ができる
```

次に，CB-80によるコンパイルのごく基本的な実行例を示します．コンパイラ「CB-80」によってリロケータブル・オブジェクト・プログラムが生成され，リンクローダ「LK80」によって各種のライブラリ「CB80．IRL」が結合され，実行可能なオブジェクト・プログラムができあがります．

図12-2-4　CB-80コンパイラの実行

```
A>CB80 SOSUCB[B]↵ ………CB-80のメイン・プログラムを起動して，ソース・プログラム「SOSUCB.BAS」
                           をコンパイルする
-----------------------------------------------
CB80 Version 1.3 Serial No. 072-0000 Copyright (c)
1981 Digital Research, Inc.  All rights reserved
-----------------------------------------------
end of compilation
no errors detected
code area size:     174        00ach
data area size:     6          0006h
common area size:   0          0000h
symbol table space remaining: 24607
コンパイル終了，コンパイル・エラーなし
A>DIR SOSUCB.*↵ …………コンパイラ実行後の「SOSUCB」ファイルを確認する

A: SOSUCB    BAS : SOSUCB    REL
   ソース・プログラム   コンパイラにより生成されたリロケータブル・
                        オブジェクト・プログラム
A>LK80 SOSUCB↵ ………「SOSUCB.REL」に対して，リンクローダ「LK80」を実行

-----------------------------------------------
LK80 Version 1.3 Serial No. 07245678 Copyright (c)
1982 Digital Research, Inc.  All rights reserved
-----------------------------------------------
code size:        1580 (0100-167F)
common size:      0000
data size:        0173 (1680-17F2)
symbol table space remaining:    95FD
リンクロード終了，リロケータブル・オブジェクト「SOSUCB.REL」と，実行時のライブラリ「CB80.IRL」が
リンクされ，実行可能なオブジェクトが生成された
A>DIR SOSUCB.*↵ ……………「SOSUCB」ファイルの確認

A: SOSUCB    BAS : SOSUCB    SYM : SOSUCB    REL : SOSUCB    COM
                   生成されたシンボル・テーブル        生成された実行可能な
                                                       オブジェクト・プログラム
A>B:STAT SOSUCB.COM↵ ………完成した実行可能なプログラム「SOSUCB.COM」の大きさを調べる

 Recs  Bytes  Ext Acc
   43     8k    1 R/W A:SOSUCB.COM ……8Kバイトである
Bytes Remaining On A: 864k

A>
```

この作業で，実行可能な純マシンコードのオブジェクト・プログラム(SOSUCB COM)ができあがりました．さっそく実行してみましょう．このプログラムは，インタープリタなどを介することなく，単独で実行できることに注目してください．

**図12-2-5　CB-80によって作成されたプログラムの実行**

```
A>SOSUCB ↲ ……………完成したプログラムの実行

10:30:16 ……………開始時刻 ┐ この間1分05秒
10:31:21 ……………終了時刻 ┘
 1229 ……………………答

A>
```

BASICインタープリタの場合に比べて，実行速度はかなり速くなり，1分05秒で同じ答が出ています．これは $N_{88}$-BASICインタープリタの約1／20の所要時間になります．このプログラムは，演算を整数で行わせていることなどの点で，実行速度に関してコンパイラが効果的に利用された例といえるでしょう．

CB-80は，BASIC言語のソース・プログラムを入力し，リロケータブル・オブジェクト・プログラムを出力しますが，その内部では，いったんアセンブリ・ソース・プログラムに変換されます．例題のBASICプログラムが，どのようなアセンブリ・ソース・プログラムに変換されているかを見てみましょう．このリストは，コンパイル時に[I]スイッチをつけることにより出力されます．

**図12-2-6　BASICプログラム→アセンブリ・ソース・プログラム(CB-80)**

```
            CALL ?INIT
1:   001ch |         INTEGER C,D,N          |
2:   001ch |         PRINT CHR$(1BH);"A"    |
            LXI  H,27
            CALL ?SCHR
            CALL ?PCSS
            LXI  H,CODE(1)
            CALL ?PCSN
3:   002bh |      C=1 : N=1      |
            LXI  H,1
            SHLD C                    DATA(0)
            LXI  H,1
            SHLD N                    DATA(4)
4:   0037h |LOOP1:                |
5:   0037h |       C=C+2 : D=2    |
            LHLD C                    DATA(0)
            INX  H
            INX  H
            SHLD C                    DATA(0)
            LXI  H,2
            SHLD D                    DATA(2)
6:   0045h |        IF C>10000 THEN PRINT CHR$(1BH);"A"  ┐
            LHLD C                    DATA(0)            └→ : PRINT N : STOP
            LXI  D,10000
```

```
          MOV  A,E
          SUB  L
          MOV  A,D
          SBB  H
          JP   @000
          LXI  H,27
          CALL ?SCHR
          CALL ?PCSS
          LXI  H,CODE(4)
          CALL ?PCSN
          LHLD N              DATA(4)
          CALL ?PCIN
          CALL ?STOP
    @000:
7:  006ah LOOP2:
8:  006ah                IF (MOD(C,D))=0 THEN GOTO LOOP1 ELSE D=D+1
          LHLD C              DATA(0)
          PUSH H
          LHLD D              DATA(2)
          CALL ?IMOD
          MOV  A,H
          ORA  L
          JNZ  @001
          JMP  LOOP1
          JMP  @002
    @001: LHLD D              DATA(2)
          INX  H
          SHLD D              DATA(2)
    @002:
9:  0086h                IF D*D > C THEN N=N+1 : GOTO LOOP1
          LHLD D              DATA(2)
          XCHG                                    ELSE GOTO LOOP2
          LHLD D              DATA(2)
          CALL ?MIDH
          XCHG
          LHLD C              DATA(0)
          MOV  A,L
          SUB  E
          MOV  A,H
          SBB  D
          JP   @003
          LHLD N              DATA(4)
          INX  H
          SHLD N              DATA(4)
          JMP  LOOP1
          JMP  @004
    @003: JMP  LOOP2
    @004: CALL ?STOP
```

参考までに，同じプログラムをマイクロソフト社のBASICコンパイラ(BASCOM)でコンパイルするためのソース・プログラムを示しておきましょう．時刻を表示するための部分がエスケープ・シーケンスになっているほかは，$N_{88}$-BASICのものと同じです．このソース・プログラムから，BASCOMで作り出されたオブジェクト・プログラムの方が，CB-80のものより実行速度が遅く，約3分ほどかかりました．しかし，このプログラムだけでは，実行速度の優劣は判断できません．

**図12-2-7　BASCOM用の例題ソース・プログラム**

```
A>TYPE SOSUMB.BAS ⏎ ………BASCOM用のソース・プログラムをタイプアウトする

100 DEFINT C,D,N
110      PRINT CHR$(&H1B);"A"
120      C=1 : N=1
130      C=C+2 : D=2
140         IF C>10000 THEN PRINT CHR$(&H1B);"A" : PRINT N : END
150            IF (C MOD D)=0 THEN GOTO 130 ELSE D=D+1
160               IF D*D > C THEN N=N+1 : GOTO 130 ELSE GOTO 150

A>       ↑ BASCOM用のソース・プログラム
```

参考までに，1～100までの間の素数は25個，1～1000までは168個，1～10000までは1229個，1～100000までは9592個となります．みなさんは，ここでの例題のような素朴なアルゴリズムのプログラムではなく，「エラトステネスのふるい」などの効果的なアルゴリズムのプログラムを作成してみてください．実行速度は劇的に改善されるでしょう．

# A

# APPENDIX

# A1 BASIC内およびCP/M内のサブルーチンの利用

本書では，次に示す3種類のプログラムを実際に作成し，それを教材として，各章でアセンブラに関するいろいろなことを解説しています．

**2章～5章　メッセージ表示プログラム**
**6章～8章　メニュー選択プログラム**
**9章～11章　メモリ・ダンプ・プログラム**

これらの例題プログラムは，機種に関係なく，なるべく多くのパーソナル・コンピュータで実習が可能なように，それぞれの機種に特有な部分を「1文字出力サブルーチン」と，「1文字キー入力サブルーチン」の2か所のみに限定して作られています．よって，この2つのサブルーチンさえ用意できれば，80系のCPUを使ったどのような機種のパーソナル・コンピュータ上でも，本書の例題を使って実習することができます．

まず，対象となる2つのサブルーチンの機能を次に示します．

- **1文字出力サブルーチン**――――Aレジスタにセットされている文字データ(アスキーコード)を，CRTディスプレイに表示する

- **1文字キー入力サブルーチン**――このルーチンが呼ばれるとキー入力待ちとなり，キー入力があれば，その文字をCRTディスプレイに表示し，かつそのデータをAレジスタにセットする

この2つのサブルーチンは，独自に作成するよりは，各機種のBASICあるいはCP/Mがその内部に持っているものを利用する方が便利です．

それぞれのシステムに用意されている各種のルーチンを使うことを，BASICの場合は「BASIC　ROM内サブルーチンコール」(BASICのROMを持たず，電源ONのたびにディスクなどからロードする機種もある)などと呼び，CP/Mの場合は「システムコール」と呼んでいます．では，これらについて解説しましょう．

## BASICのROM内サブルーチンコール

80系のCPUを使った，代表的なパーソナル・コンピュータの，BASIC内の各種サブルーチンの中から，本書で必要とする1文字出力サブルーチンと，1文字キー入力サブルーチンのエントリー・ポイント(入口アドレス)を示します．CP/M上ではなく，BASIC上で各例題プログラムを実習する場合は，この表を利用して，それぞれの機種用のアドレスにソース・プログラムを変更してください(本書のリストにあるのは，PC-8801の $N_{88}$-BASIC用のアドレスです)．

| 機　種 | 1文字入力 | | 1文字出力 | |
|---|---|---|---|---|
| | エントリ | パラメータ | エントリ | パラメータ |
| PC-8801(mkIIを含む) $N_{88}$-BASIC | 3583H | 入力データ→Aレジスタ | 3E0DH | 出力データ→Aレジスタ |
| PC-8001(mkIIを含む)<br>PC-8801(mkIIを含む) N-BASIC | 0F75H | 入力データ→Aレジスタ | 0257H | 出力データ→Aレジスタ |
| PC-6001(mkIIを含む)<br>PC-6601 $N_{60}$-BASIC | 0FC4H | 入力データ→Aレジスタ | 1075H | 出力データ→Aレジスタ |
| MSX MSX-BASIC | 009FH | 入力データ→Aレジスタ | 00A2H | 出力データ→Aレジスタ |
| PASOPIA(5, 7を含む) T-BASIC | 065DH | 入力データ→Aレジスタ | 0892H | 出力データ→Aレジスタ |
| X1(C, D, turboを含む) Hu-BASIC | 001BH | 01→Aレジスタとして呼び出すと，入力データ→Aレジスタ | 0013H | 出力データ→Aレジスタ |

図A-1-1　機種別の各サブルーチンのエントリーポイント

# CP/Mのシステムコール

次に，CP/Mのシステムコールについて，そのあらましを解説しておきましょう．BASICのROM内コールの場合は，例えば$N_{88}$-BASICでは，1文字出力サブルーチンは$3E0D_H$，1文字キー入力サブルーチンは$3583_H$，というように，それぞれの機能により，それらを利用するためのエントリー・ポイントが異なります．機種が異なればなおさらです．

これに対してCP/Mは，機種や機能に関係なく，すべての場合についてのエントリー・ポイントが0005番地に固定されています．そのかわり，機能別につけられた「ファンクションNo.」(機能番号)をCレジスタにセットして 0005番地をコールします．このCレジスタにセットされる値により，入口は1か所でも，その内部で自動的にそれぞれの機能のルーチンへ分岐されるようになっています．

1文字出力サブルーチンの場合は，その機能番号が2なので，

このような具合になります．例えば，Eレジスタにセットする「文字データ」が$41_H$であれば"A"，$35_H$であれば"5"というように，CRTディスプレイ上に表示されます．

システムコールは，この例のように，各ファンクションに引き渡すパラメータがある場合には，EレジスタあるいはDEレジスタペアにそのパラメータをセットしてから，0005番地をコールします．

次に，1文字キー入力サブルーチンの例を示しましょう．1文字キー入力サブルーチンの機能番号は1なので，

となります．この場合にはサブルーチンに引き渡すパラメータはなく，キー入力があれば，そのデータがAレジスタにセットされてこのサブルーチンから帰ります．

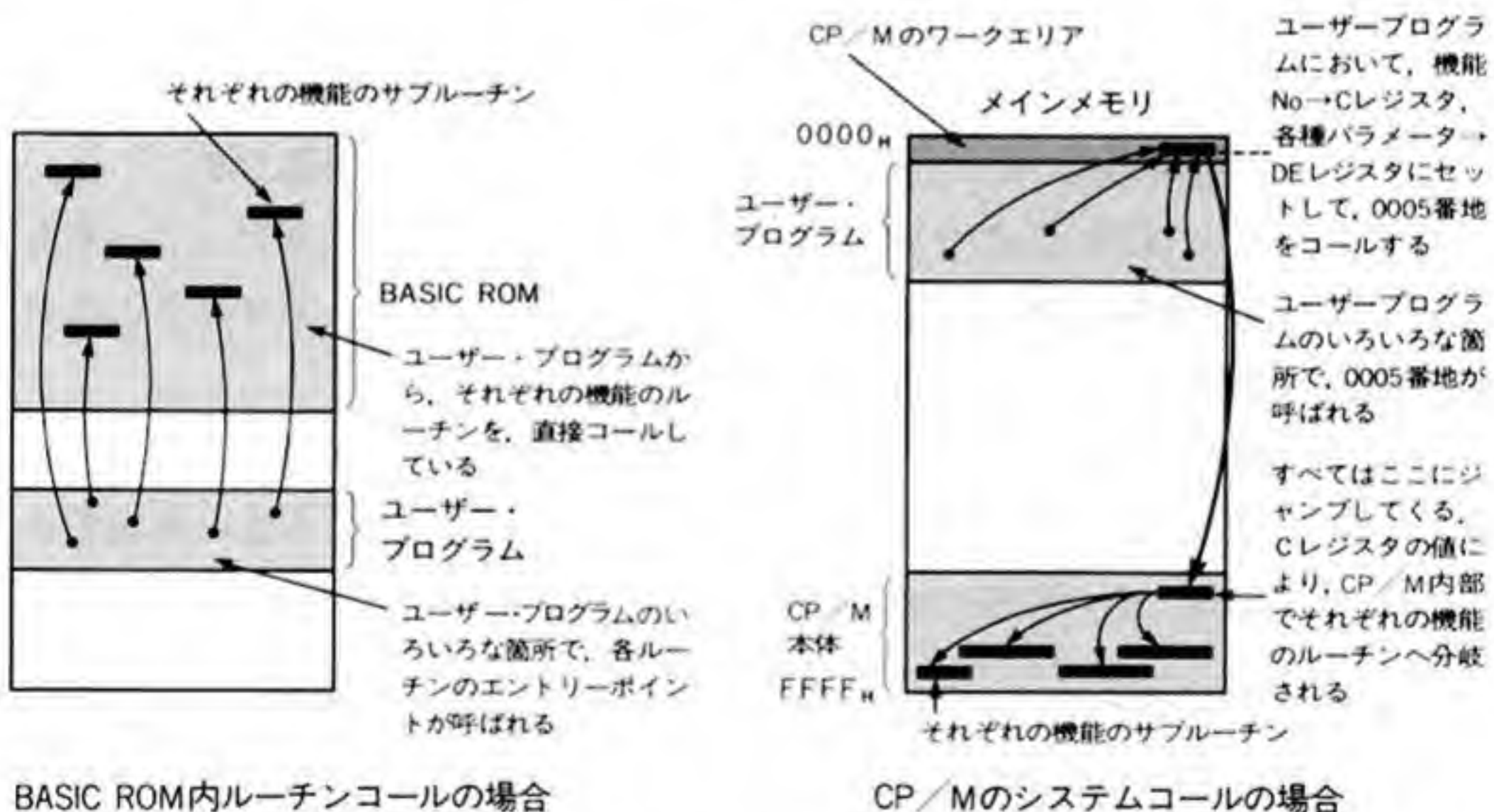
もう一例，2～5章でおなじみの，メッセージ表示プログラムを，システムコールを使って作成してみましょう．ファンクションNo.9に，文字列の出力機能が用意されていますので，次に示すような非常に簡単なプログラムになります．

```
BDOS    EQU     0005H
CR      EQU     0DH
LF      EQU     0AH

        ORG     100H
        LD      C , 9
        LD      DE , MESG
        CALL    BDOS
        RET

MESG :  DB      CR , LF , 'Good  Morning' , CR , LF , '$'
        END
```

このように，Cレジスタに9をセットし，DEレジスタペアに文字列エリアの先頭アドレスをセットして，0005番地をコールするだけでよいのです．ただし，文字列の最後に必ず[＄]を置いてください．この[＄]が，2～5章の例題プログラムの「EOS」に相当し，文字列の終りを示すマークとなります．

CP／Mマシンで実習されている方は，各自でこのプログラムの動作を確認してみてください．

CP／Mでは，このようにして任意のファンクションのサブルーチンをコールするわけですが，このファンクションの種類は，CP／M version 2.2の場合は約40ほどあり，これらの機能がユーザーの利用を目的として体系化され，積極的に提供されています．

ところがBASICの場合は，ほとんどの機種が，BASIC内ルーチンのユーザー・プログラムからの使用を公開しておらず，ユーザー側が勝手に利用しているのに過ぎません．これは，各システム内のルーチンの利用について，根本的に考え方の違う点です．次の図に，BASICのROM内ルーチンのコールと，CP／Mのシステムコールの呼び出し方の違いを，対比して示しておきましょう．なお，CP／Mのシステムコールについては，拙著『応用CP／M』で，例題プログラムを使って詳しく解説していますので参照してください．

**図A-1-2　BASICのROM内ルーチンコールとCP／Mのシステムコールの概念**

# A2 インテルHEX形式のオブジェクト・プログラムについて

アセンブラやリンクローダ，それに各種のコンパイラなどから生成されるオブジェクト・プログラムの形式は，次の3つが代表的であり，これらは本書でも何度か登場しています．

1．インテルHEX形式
2．マイクロソフト・リロケータブル・オブジェクト形式
3．実行可能形式(純マシンコード形式)

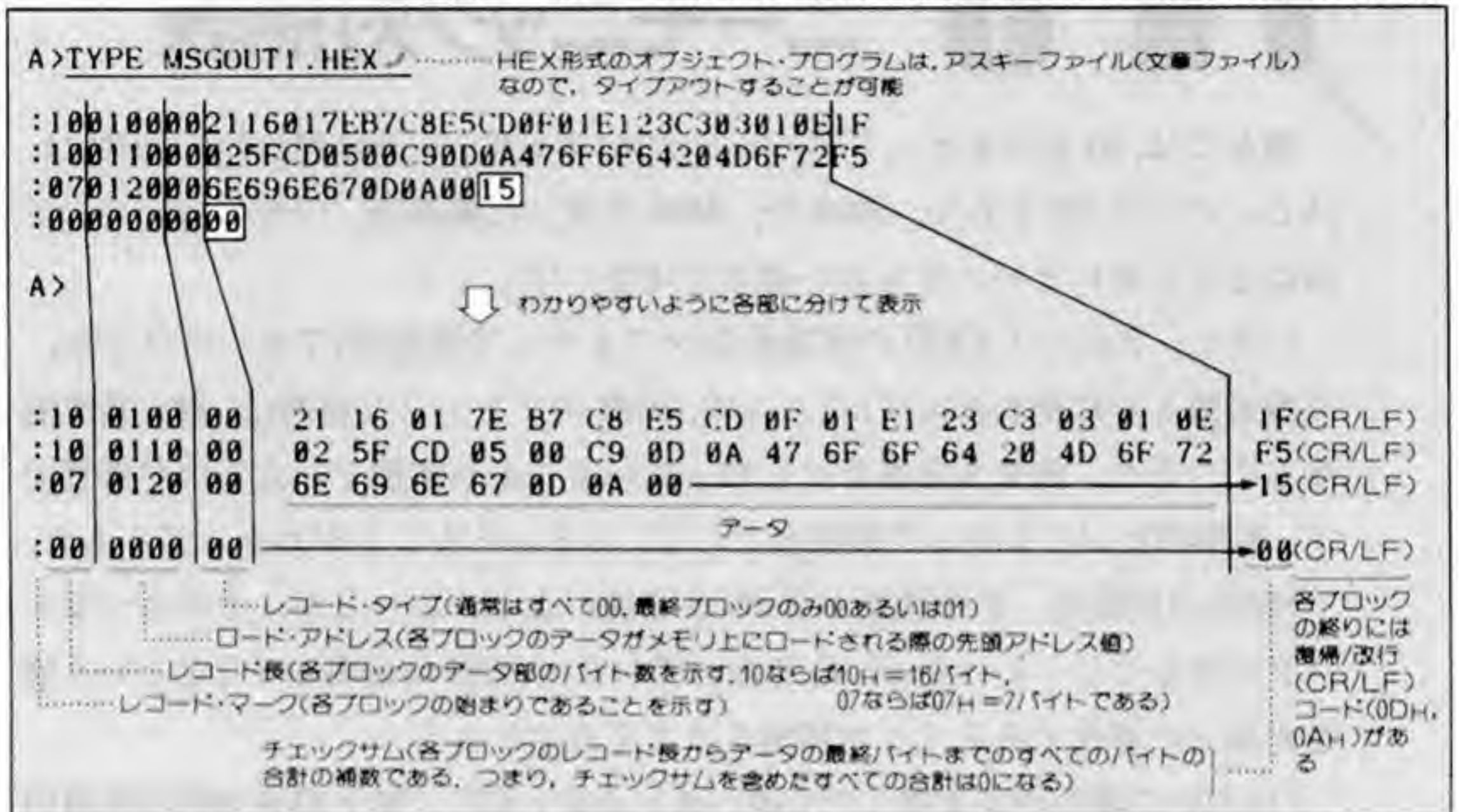
最終的にコンピュータのメモリ上にロードされ(あるいはROM上に固定され)てそのプログラムが実行されるのは，3番目の実行可能形式ですが，これに至るまでの開発過程では，1と2のオブジェクト形式が主役です．

特にインテルHEX形式のオブジェクト・プログラムは最もポピュラーであり，アセンブル作業に使われるだけでなく，オブジェクト・プログラムの交換や通信の手段としても広く使われています．

これは，構成が単純であるにもかかわらず，インテルHEX形式が，オブジェクト・プログラムを表現する一般的な形式として，非常にうまい方式であるからでしょう．その特長のいくつかを次に示します．

- どのようなオブジェクト・プログラムでも，次の限定された文字，「0～F」，「：」，「復帰コード(0DH)」，「改行コード(0AH)」だけを使用した文字列として表現するので，データの受渡しが容易であり，かつCRTディスプレイなどで「読む」ことが可能である
- 任意のブロック単位に分割することができ，それぞれにチェックサムを付加できるので，データ受渡しの確認が容易である
- オブジェクト・プログラムのロード・アドレスが，ブロック単位で付加されている

では，本書の2～5章でおなじみの，メッセージ表示プログラムの作成過程で生成された，インテルHEX形式のオブジェクト・プログラムを例に，その構成について解説しましょう．

図A-2-1(a)　インテルHEX形式のオブジェクト・プログラムの構成例

もう一例を示しましょう．表現は多少違いますが(レコード長が異なる)，内容はまったく同じオブジェクト・プログラムです．

図A-2-1(b)　インテルHEX形式のオブジェクト・プログラムの構成例

```
A>TYPE MSGOUT2.HEX
:1C010000002116017EB7C8E5CD0F01E123C303010E025FCD0500C90D0A476F6F6477
:0B011C00204D6F726E696E670D0A00C7      チェックサム
:0000000000                              チェックサム
A>

              (今回のものは，このブロックは1CH＝28バイトである)
:1C 0100 00   21 16 01 7E B7 C8 E5 CD 0F 01 E1 23 C3 03 01 0E 02 5F→ *
                    *→CD 05 00 C9 0D 0A 47 6F 6F 64              77
              (このブロックは0BH＝11バイトである
:0B 011C 00   20 4D 6F 72 6E 69 6E 67 0D 0A 00                   C7
:00 0000 00                                                      00
                              データ
レコード・タイプ
ロード・アドレス                                         チェックサム
レコード長
レコード・マーク
```

# A3 8080対Z-80 ニーモニック対照表

現在では，80系の8ビットのパーソナル・コンピュータに使われるCPUは，ほとんどがZ-80であり，8080や，8085を使った製品は，「パソコン」としてはほとんどないといっても言い過ぎではないでしょう．

しかし，8ビットCPUの本格的なソフトウェア開発者(アセンブラとか，高級言語とかにかかわらず)にとって，8080のアセンブリ言語は，最も基本的なものであり，何をする場合にも知っていることが前提といえるほど重要です．本格的なソフトウェア開発者として，いろいろなことがわかってくると，「8080」の位置は，その機能がZ-80より低いとか，古いとか，そのニーモニックが分かりにくいとか，Z-80を知っていれば8080は必要ないなどという議論を超えた存在であることが理解されてくるでしょう．

『はじめて読むマシン語』でも述べましたが，CP／M上の各分野の本格的なソフトウェアの多くは，Z-80専用ではなく，8080用に作られています．Z-80は，8080の機能をそっくり含み，その上に多くの機能を拡張したCPUですので(このことを，8080の上位コンパチブルと呼ぶ)，私たちは，8080用に作られたソフトウェアであっても，そのことを気にすることなく，そのままZ-80マシンで実行しているわけです．

一例を挙げるなら，ビジネスソフトでは，ワードスター，スーパーカルク，マルチプラン，dBASE II…，言語ソフトでは，FORTRAN-80，BDS C，LSI C，Pascal MT＋などの代表的なソフトウェア製品のほとんどは，8080用であり，Z-80専用のものは，数えるほどしかありません．本書に登場する8080，Z-80アセンブラの代表である「MACRO-80」もそのプログラム自身は，8080の命令だけでできています(つまり，Z-80のソース・プログラムをアセンブルする場合でも，8080マシン上でアセンブルが可能)．

特に，言語ソフトの各種コンパイラでは，オブジェクト・プログラムを生成する前段階として，それぞれの言語のソース・プログラムが，アセンブリ・ソース・プログラムに展開されるものがありますが，それらはほとんどが 8080 のアセンブリ・ソース・プログラムに展開されます．よって各種の言語のユーザーにとって，8080 アセンブリ言語の知識は必須といってもよいでしょう．

また現在，16 ビット CPU で最も多く使われている 8086, 8088 は，8 ビット CPU の 8080, 8085 のユーザーが容易に 16 ビットへ移行できるように，その延長線上にあります(開発は同じインテル社，あとがき参照)．よって，8086, 8088 のアセンブラを学ぼうとする人は，8080 のアセンブリ言語を知っている方が，断然有利です．

このようなことから，本格的なソフトウェア開発者としては，8080 のアセンブリ言語を，身につけておくことをお勧めします．

ここでは，CP/Mの 8080 アセンブラなどを使用する人のために，8080 の全インストラクション(同じパターンのものは代表して1つのレジスタの例を示し，その他のバリエーションは省略してある)に対する，Z-80 のインストラクションを示しておきましょう．

図A-3-1　8080対Z-80ニーモニック対照表

| オブジェクトコード(マシン語) | 8080ニーモニック | Z-80ニーモニック | 備　考 |
|---|---|---|---|
| 78 | MOV B,C | LD B,C | ここでのBレジスタおよびCレジスタは各レジスタを代表したものであり，それぞれが，A，B，C，D，E，H，Lレジスタのいずれかと置き換わる命令の種類がある |
| 3E12 | MVI B,12H | LD B,12H | |
| 77 | MOV M,B | LD (HL),B | |
| 3612 | MVI M,12H | LD (HL),12H | |
| 3C | INR B | INC B | |
| 3D | DCR B | DEC B | |
| 34 | INR M | INC (HL) | |
| 35 | DCR M | DEC (HL) | |
| 87 | ADD B | ADD A,B | |
| 8F | ADC B | ADC A,B | |
| 97 | SUB B | SUB B | |
| 9F | SBB B | SBC A,B | |
| A7 | ANA B | AND B | |
| AF | XRA B | XOR B | |
| B7 | ORA B | OR B | |
| BF | CMP B | CP B | |
| 86 | ADD M | ADD A,(HL) | ここでのBレジスタおよびBCレジスタは，各レジスタを代表したものであり，8080の場合はそれぞれ，B，D，H，SP，Z-80の場合はBC，DE，HL，SP，レジスタのいずれかと置き換わる命令の種類がある |
| 8E | ADC M | ADC A,(HL) | |
| 96 | SUB M | SUB (HL) | |
| 9E | SBB M | SBC A,(HL) | |
| A6 | ANA M | AND (HL) | |
| AE | XRA M | XOR (HL) | |
| B6 | ORA M | OR (HL) | |
| BE | CMP M | CP (HL) | |
| C612 | ADI 12H | ADD A,12H | |
| CE12 | ACI 12H | ADC A,12H | |
| D612 | SUI 12H | SUB 12H | |
| DE12 | SBI 12H | SBC A,12H | |
| E612 | ANI 12H | AND 12H | |
| EE12 | XRI 12H | XOR 12H | |
| F612 | ORI 12H | OR 12H | |
| FE12 | CPI 12H | CP 12H | |
| 07 | RLC | RLCA | |
| 0F | RRC | RRCA | |
| 17 | RAL | RLA | |
| 1F | RAR | RRA | |

| オブジェクトコード<br>(マシン語) | 8080ニーモニック | Z-80ニーモニック | 備　考 |
|---|---|---|---|
| 27 | DAA | DAA | ここでのBレジスタおよびBCレジスタは，各レジスタを代表したものであり，8080の場合はそれぞれ，B，D，H，SP，Z-80の場合はBC，DE，HL，SP，レジスタのいずれかと置き換わる命令の種類がある |
| 2F | CMA | CPL | |
| 37 | STC | SCF | |
| 3F | CMC | CCF | |
| 02 | STAX　B | LD　(BC),A | |
| 0A | LDAX　B | LD　A,(BC) | |
| 12 | STAX　D | LD　(DE),A | |
| 1A | LDAX　D | LD　A,(DE) | |
| 323412 | STA　1234H | LD　(1234H),A | |
| 3A3412 | LDA　1234H | LD　A,(1234H) | |
| 013412 | LXI　B,1234H | LD　BC,1234H | |
| 09 | DAD　B | ADD　HL,BC | |
| 03 | INX　B | INC　BC | |
| 0B | DCX　B | DEC　BC | |
| 223412 | SHLD　1234H | LD　(1234H),HL | |
| 2A3412 | LHLD　1234H | LD　HL,(1234H) | |
| E3 | XTHL | EX　(SP),HL | |
| EB | XCHG | EX　DE,HL | |
| E9 | PCHL | JP　(HL) | |
| F9 | SPHL | LD　SP,HL | |
| F5 | PUSH　PSW | PUSH　AF | |
| C5 | PUSH　B | PUSH　BC | |
| D5 | PUSH　D | PUSH　DE | |
| E5 | PUSH　H | PUSH　HL | |
| F1 | POP　PSW | POP　AF | |
| C1 | POP　B | POP　BC | |
| D1 | POP　D | POP　DE | |
| E1 | POP　H | POP　HL | |
| C33412 | JMP　1234H | JP　1234H | |
| C23412 | JNZ　1234H | JP　NZ,1234H | |
| CA3412 | JZ　1234H | JP　Z,1234H | |
| D23412 | JNC　1234H | JP　NC,1234H | |
| DA3412 | JC　1234H | JP　C,1234H | |
| E23412 | JPO　1234H | JP　PO,1234H | |
| EA3412 | JPE　1234H | JP　PE,1234H | |
| F23412 | JP　1234H | JP　P,1234H | |
| FA3412 | JM　1234H | JP　M,1234H | |

| オブジェクトコード（マシン語） | 8080ニーモニック | Z-80ニーモニック | 備　考 |
|---|---|---|---|
| CD3412 | CALL　1234H | CALL　1234H | ここでのBレジスタおよびBCレジスタは，各レジスタを代表したものであり，8080の場合はそれぞれ，B，D，H，SP，Z-80の場合はBC，DE，HL，SP，レジスタのいずれかと置き換わる命令の種類がある |
| C43412 | CNZ　1234H | CALL　NZ,1234H | |
| CC3412 | CZ　1234H | CALL　Z,1234H | |
| D43412 | CNC　1234H | CALL　NC,1234H | |
| DC3412 | CC　1234H | CALL　C,1234H | |
| E43412 | CPO　1234H | CALL　PO,1234H | |
| EC3412 | CPE　1234H | CALL　PE,1234H | |
| F43412 | CP　1234H | CALL　P,1234H | |
| FC3412 | CM　1234H | CALL　M,1234H | |
| C9 | RET | RET | |
| C0 | RNZ | RET　NZ | |
| C8 | RZ | RET　Z | |
| D0 | RNC | RET　NC | |
| D8 | RC | RET　C | |
| E0 | RPO | RET　PO | |
| E8 | RPE | RET　PE | |
| F0 | RP | RET　P | |
| F8 | RM | RET　M | |
| D312 | OUT　12H | OUT　(12H),A | |
| DB12 | IN　12H | IN　A,(12H) | |
| F3 | DI | DI | |
| FB | EI | EI | |
| C7 | RST　0 | RST　0 | |
| CF | RST　1 | RST　8 | |
| D7 | RST　2 | RST　10H | |
| DF | RST　3 | RST　18H | |
| E7 | RST　4 | RST　20H | |
| EF | RST　5 | RST　28H | |
| F7 | RST　6 | RST　30H | |
| FF | RST　7 | RST　38H | |
| 76 | HLT | HALT | |
| 00 | NOP | NOP | |

# A4 アスキーコード一覧表

コンピュータのデータとして，文字や文字列を扱う場合，それらはオブジェクト・プログラム中では，次の表にあるアスキーコードに変換されて取り扱われます．

本書の例題プログラムで使われる，1文字出力サブルーチンや1文字キー入力サブルーチンも，このアスキーコードでデータのやり取りが行われます．

上位→ 下位↓

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | $D_E$ | | 0 | @ | P | | p | ▁ | ┴ | | ー | タ | ミ | ＿ | × |
| 1 | $S_H$ | $D_1$ | ! | 1 | A | Q | a | q | ▂ | ┬ | 。 | ア | チ | ム | ╞ | 円 |
| 2 | $S_X$ | $D_2$ | " | 2 | B | R | b | r | ▃ | ┤ | 「 | イ | ツ | メ | ╪ | 年 |
| 3 | $E_X$ | $D_3$ | # | 3 | C | S | c | s | ▄ | ├ | 」 | ウ | テ | モ | ╡ | 月 |
| 4 | $E_T$ | $D_4$ | $ | 4 | D | T | d | t | ▅ | ▔ | 、 | エ | ト | ヤ | ◢ | 日 |
| 5 | $E_Q$ | $N_K$ | % | 5 | E | U | e | u | ▆ | ─ | ・ | オ | ナ | ユ | ◣ | 時 |
| 6 | $A_K$ | $S_N$ | & | 6 | F | V | f | v | ▇ | │ | ヲ | カ | ニ | ヨ | ◥ | 分 |
| 7 | $B_L$ | $E_B$ | ' | 7 | G | W | g | w | █ | ▕ | ァ | キ | ヌ | ラ | ◤ | 秒 |
| 8 | $B_S$ | $C_N$ | ( | 8 | H | X | h | x | ▏ | ┌ | ィ | ク | ネ | リ | ♠ | |
| 9 | $H_T$ | $E_M$ | ) | 9 | I | Y | i | y | ▎ | ┐ | ゥ | ケ | ノ | ル | ♥ | |
| A | $L_F$ | $S_B$ | * | : | J | Z | j | z | ▍ | └ | ェ | コ | ハ | レ | ◆ | |
| B | $H_M$ | $E_C$ | + | ; | K | [ | k | { | ▌ | ┘ | ォ | サ | ヒ | ロ | ♣ | |
| C | $C_L$ | → | , | < | L | ¥ | l | ¦ | ▋ | ╭ | ャ | シ | フ | ワ | ● | |
| D | $C_R$ | ← | - | = | M | ] | m | } | ▊ | ╮ | ュ | ス | ヘ | ン | ○ | |
| E | $S_O$ | ↑ | . | > | N | ^ | n | ~ | ▉ | ╰ | ョ | セ | ホ | ゛ | ╱ | |
| F | $S_I$ | ↓ | / | ? | O | _ | o | | ┼ | ╯ | ッ | ソ | マ | ゜ | ╲ | |

(スペース)20H　41H　61H　B1H

ブザー 07H　改行 0AH　復帰 0DH

5AH　7AH　DDH

このエリアのものは「文字」ではなく，コントロール・キャラクタと呼ばれる各種の制御用に使われるコードである

このエリアのものは，それぞれのパーソナル・コンピュータによって異なる．多くはここにグラフィック・キャラクタなどを割りあてている．これはPC-8001，PC-8801のもの

図A-4-1　アスキーコード一覧表

# あとがき

8ビットのパーソナル・コンピュータに使われているCPUの多くは，ザイログ社のZ-80とその同等品（NECの$\mu$PD780などのことで，セカンドソースとも呼ばれる）です．インテル社の8ビットCPUの8080や8085は，パソコン・ユーザーからは忘れられた存在であるかのようです．

確かに機能的には，Z-80は8080の機能をすべて含み，そのオブジェクトコードもまったく同じで，その上に多くの機能が拡張されているのですから，Z-80に置き換えられてしまうのも当然でしょう．しかし，これは8ビットの世界だけを見た場合のことであり，16ビットや，これからの32ビットCPUを発展的に考えた場合，このような局所的な見方はできません．そこにはインテル社の，

| [8ビット] | | [16ビット] | | | | | | [32ビット] |
|---|---|---|---|---|---|---|---|---|
| 8080，8085 | ⇨ | 8086<br>(8088) | ⇨ | 80186<br>(80188) | ⇨ | 80286 | ⇨ | 80386 |

という大河のような流れがあります．8086，8088は，PC-9801や，IBM-PCでおなじみの現在の16ビット・パーソナル・コンピュータの主流CPUです．80186はFM-16$\beta$で，80286はIBM-PC ATで使われ始めましたが，これから急速に普及していくでしょう．そしてその先は32ビットの80386へとつながっていきます．そしてこれらはいずれも，パーソナル・コンピュータの主流CPUとなる可能性が非常に大きいのです．

このように，パーソナル・コンピュータのソフトウェアの源流は，8ビット CPU の 8080 から発しています．流れのそれぞれの段階では，下位の CPU の知識を基にして，さらに高度な知識が積み上げられています．1章で，8ビット CPU がコンピュータの原点であることを強調したのも，APPENDIX 3で，8080 アセンブラは，何をするにも知っていることが前提‥‥と述べたのも，このためなのです．本書は，このことを考慮して，8080 アセンブラによる実行例も併記し，8080 対 Z-80 のニーモニック対照表を APPENDIX 3 に示しています．

いつの日か，16 ビットの 8086 や 80186，80286 に進んでいこうとする読者は，この機会に，Z-80 だけでなく，8080 アセンブラをぜひ習得しておくことをお勧めします．世の中が，たとえ 80386CPU を搭載した 32 ビット・スーパー・パーソナル・コンピュータ全盛の時代になったとしても，その知識の源は，『はじめて読むマシン語』や，本書『はじめて読むアセンブラ』で語られている 8 ビット CPU の基礎知識であり，すべての知識はここから流れ出すことに変わりはないのです．

このことを踏まえ，今後はさらにいくつもの情報を集め，ゆたかな基礎知識を土台として，次の目標へ挑戦してください．

# 索　　引

## 参考文献

はじめて読むマシン語
入門 CP/M
実習 CP/M
応用 CP/M
(いずれも村瀬康治著，アスキー出版局発行)

## はじめて読むアセンブラ

1985年3月25日　初版発行
1989年9月21日　第1版第9刷発行
定価1,650円(本体1,602円)

著　者　村瀬康治
発行者　塚本慶一郎
発行所　株式会社アスキー
〒107 東京都港区南青山6-11-1スリーエフ南青山ビル
振　替　東京4-161144
TEL (03)486-7111(大代表)
情報TEL (03)498-0299(ダイヤルイン)
出版営業部TEL (03)486-1977(ダイヤルイン)



編集担当　土田米一・大江朋子
表紙担当　郷　啓子
印刷　モリモト印刷株式会社

ISBN4-87148-774-1 C3055 P1650E

# の書籍。好評発売中！

## はじめて読むマシン語

村瀬康治著　定価1,240円（本体1,204円）

マシン語を理解するには、マシン語学習以前の基礎的な知識を学ばなければなりません。本書は、コンピュータの仕組みやCPUの働きなどの基礎を説明したあとで、実際のマシン語プログラミングについて解説しています。本書を読めば、初心者の方でもすんなりマシン語をマスターできるでしょう。PC-8801、PC-8001、X1などZ80、8080系マシンユーザー必携。

## はじめて読むBASIC

高玉皓司著　定価1,240円（本体1,204円）

これからパソコンをはじめようとする人のために、BASICの使い方をやさしく解説。機械用語・電気用語、その他もろもろのカタカナ語は極力排除するように配慮してありますので、どなたでも気軽に読んでいただける内容になっています。
内容：ふれること慣れること〔体験編〕／知るべきことを知る〔基礎編〕／できることいろいろ〔中級編〕／表現力をつける〔グラフィック編〕／使いこなすために〔上級編〕

## はじめて読む8086

村瀬康治監修　蒲地輝尚著　定価1,650円（本体1,602円）

多くの16ビット・コンピュータで使われている8086、V30系CPUの仕組みをわかりやすく解説。本書では、MS-DOS上のコマンドを使って実習を行っており、どのマシンでも同じように学習を進めていくことができます。また本書での解説は、MS-DOSやアセンブラ、C言語などを学ぶ上での基礎知識となっていますから、これから本格的に学習をしたい方にも最適な一冊です。
内容：マシン語から広がる世界／実行型ファイルをダンプする　実行型ファイルのメッセージを変更する／これだけは覚えて欲しいコンピュータの知識／8086CPUの基礎 etc.

# はじめて読むシリーズ

## はじめて読む8086

蒲地輝尚 著　村瀬康治 監修　定価1,650円(本体1,602円)

多くの16ビット・コンピュータで採用されている8086, V30, 80286系CPUのマシン語を, MS-DOSの標準ツールを使ってやさしく実習. これからMS-DOSやアセンブラの上級にチャレンジしようとする読者には必須の書籍です.

## はじめて読むマシン語

村瀬康治 著　定価1,240円 (本体1,204円)

はじめてマシン語を学ぶ人のための啓蒙的入門書. コンピュータの基礎知識もあわせて解説していますから, 初心者でも十分に読みこなすことができます. PC-8801シリーズ, X1シリーズ, MSXなどZ80, 8080系マシンユーザー必携.

MACHINE LANGUAGE

## はじめて読む6809

星山浩樹 著　村瀬康治 監修　定価1,440円(本体1,398円)

本書は, 非常にわかりやすいと大好評の「はじめて読むマシン語」の6809版です. やさしい解説と豊富なサンプルを読み進めるうちに, 自然にマシン語が身につけられます. FM-77シリーズ, レベル3シリーズなどのユーザーに最適.

## はじめて読むBASIC

高玉皓司 著　定価1,240円(本体1,204円)

これからパソコンをはじめようとする人のために, BASICの使い方をやさしく解説. 機械用語・電気用語など, もろもろのカタカナ語は極力排除するように配慮してありますので, どなたでも手軽に読んでいただける内容となっています.

BASIC